Verteilte Nutzungskontrolle und Provenance Tracking am Beispiel von Cloud-Technologien

Bachelorarbeit

KIT – Karlsruher Institut für Technologie
Fraunhofer IOSB – Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung

Nicolas Schuler

21. Dezember 2020

Verantwortlicher Betreuer: Prof. Dr.-Ing. habil. Jürgen Beyerer
Betreuende Mitarbeiter: Christian Kühnle, M.Sc.
Paul Georg Wagner, M.Sc.

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung beachtet habe.

Karlsruhe, den 21. Dezember 2020

(Nicolas Schuler)

Zusammenfassung

In den letzten Jahrzehnten haben sich Daten zu einem der wertvollsten Rohstoffe entwickelt. Im Zuge dessen werden Forderungen nach einem transparenten und kontrollierbaren Umgang mit Daten immer lauter.

Nutzungskontrolle und Provenance Tracking sind Konzepte, mit denen diese Forderungen umgesetzt werden können. Nutzungskontrolle ergänzt hierzu die Zugriffskontrolle um den Umstand, dass die Nutzung von Daten – auch nachdem sie verbreitet wurden – kontrolliert werden kann. Provenance Tracking dient hingegen zur Bestimmung der Herkunft eines Datums. Das für Nutzungskontrolle und Provenance Tracking notwendige Fundament – die Infrastruktur – könnte hierbei durch Cloud-Technologien, wie Kubernetes, bereitgestellt werden.

Dahingehend bietet diese Arbeit einen Überblick über den aktuellen Forschungsstand zu Nutzungskontrolle und Provenance Tracking. Dies umfasst insbesondere mögliche Bezüge zum Themengebiet *Cloud Computing* und aktuelle Forschungsprojekte, wie International Data Spaces (IDS) und deren Referenzarchitektur. Des Weiteren wird ein grundlegendes Verständnis für den Begriff Cloud geschaffen. Im Besonderen wird auf die Aspekte Sicherheit und Recht im Kontext von *Cloud Computing* eingegangen. Schlussendlich werden die dadurch gewonnen Erkenntnisse zur Ausbringung einer prototypischen Nutzungskontroll- sowie Provenance sammelnden Infrastruktur genutzt. Die Cloud-Technologie Kubernetes sowie hierfür entwickelte *Dummy*-Komponenten der Referenzarchitektur bilden das Fundament dieser Infrastruktur. Die Modellierung und Implementierung ist daraufhin Gegenstand einer Evaluation und Diskussion mit Fokus auf operativen und sicherheitsrelevanten Aspekten.

Inhaltsverzeichnis

1. Einleitung

Der Artikel *The world's most valueable resource is no longer oil, but data* der englischsprachigen Wochenzeitung *The Economist* sorgte im Jahr 2017 für viel Diskussionsstoff [Eco17]. Die eigentliche Aussage *Data is the new oil* geht auf ein Zitat des britischen *Data Scientists* Clive Humby aus dem Jahre 2006 zurück. Während ein Großteil der Diskussion auf kartellrechtliche Belange abzielte, ließe sich im Kontrast dazu eine Vielzahl an Verbesserungen anführen, die auf den Perspektivenwechsel im Umgang mit Daten zurückzuführen sind. Selbst Öl war und ist immer noch für Verschmutzungen, Konflikte und vieles mehr verantwortlich. Dennoch leitete das schwarze Gold ein neues Zeitalter ein, das der gesamten Menschheit neue Möglichkeiten eröffnete [Bha19].

Die Wichtigkeit von Daten spiegelt sich auch in diversen Vorhersagen großer Unternehmensberatungen, wie Gartner und *McKinsey & Company* wider. Gartner sagt hierzu voraus, dass im Jahr 2022 etwa 90% der Unternehmensstrategien Daten als kritische Unternehmenswerte und *Data Analytics* als unentbehrliche Kompetenz aufschlüsseln werden. Darüber hinaus sollen 35% der großen Unternehmen als Käufer oder Verkäufer an sogenannten *Data Marketplaces* aktiv sein, im Gegensatz zu den bereits 25% im Jahr 2020. Umso besorgniserregender ist die Prognose, dass bis 2022 die korrekte Identifizierung von vertrauenswürdigen Daten und Quellen bei unter 5% liegen soll. Dahingehend wird es immer wichtiger, entsprechende Infrastruktur und Mechanismen vorweisen zu können, die gegenseitiges Vertrauen fördern und den Datenaustausch transparent und kontrollierbarer machen. Glaubt man den Prognosen von Gartner, so wird *Cloud Computing* diese Infrastruktur sein, denn im Jahr 2024 sollen mehr als 45% der IT-Ausgaben an *Cloud Computing* entfallen. Bereits im Jahr 2022 sollen *Public Cloud Services* für 90% der Innovationen im Bereich *Data and Data Analytics* verantwortlich sein. Daher ist es auch nicht verwunderlich, dass Unternehmen sich bereits jetzt in Richtung Cloud aufstellen. So kaufte nicht zuletzt IBM den Softwarehersteller RedHat auf [Bri19] und die Deutsche Bahn (DB) wechselte von ihren (in Spitzenzeiten) bis zu 8000 selbstbetriebenen Servern zu Cloud-Lösungen von Amazon AWS und Microsoft Azure [Der20; Gartner20A; Gartner20B; Gartner20C].

Nicht nur aus wirtschaftlicher Sicht ist ein transparenter und kontrollierbarer Datenaustausch wünschenswert. Seit Inkrafttreten der Datenschutz-Grundverordnung (DSGVO) ergibt

sich dafür eine rechtliche Notwendigkeit, sollten personenenbezogene Daten verarbeitet werden.

Mit Nutzungskontrolle und Provenance Tracking existieren Konzepte, durch die ein transparenter, kontrollierbarer und vertrauensvoller Datenaustausch möglich wäre. Dazu ergänzt Nutzungskontrolle die klassischen Konzepte der Zugriffskontrolle um den Umstand, dass die Nutzung von Daten – auch nachdem sie verbreitet wurden – kontrolliert werden kann. Die Klärung der Herkunft eines Datums obliegt dem Provenance Tracking. Darüber hinaus kann aus den oben genannten Gründen die Cloud und deren Technologien als zukünftiges Fundament für den Datenaustausch betrachtet werden. Wie dieses Fundament aussehen kann und inwiefern Nutzungskontrolle und Provenance Tracking darin integriert werden können, ist Gegenstand der vorliegenden Arbeit.

1.1. Zielsetzung

Das Ziel der Arbeit ist es, zunächst einen aktuellen Überlick über die Konzepte der Nutzungskontrolle und des Provenance Trackings zu geben. Mögliche Bezüge zum Themengebiet *Cloud Computing* sollen dabei herausgestellt werden. Ferner soll eine Beschreibung der Referenzarchitektur zur Nutzungskontrolle und des Provenance Trackings im International Data Spaces (IDS) stattfinden. Anschließend erfolgt eine Auseinandersetzung mit dem Themengebiet *Cloud Computing*. Dahingehend soll insbesondere auf die Aspekte Sicherheit und Recht eingegangen werden. Die gesammelten Erkenntnisse werden daraufhin zur Ausbringung einer prototypischen Nutzungskontroll- sowie Provenance sammelnden Infrastruktur genutzt, welche mittels *Dummy*-Komponenten aus der Referenzarchitektur und der Cloud-Technologie Kubernetes umgesetzt wird. Die Modellierung sowie Implementierung ist daraufhin Gegenstand einer Evaluation und Diskussion.

1.2. Struktur der Arbeit

Die vorliegende Arbeit gliedert sich in fünf Kapitel. In Kapitel 2 wird der aktuelle Stand von Nutzungskontrolle sowie Provenance Tracking zusammengefasst, wobei auf Überschneidungen mit dem Themengebiet *Cloud Computing* eingegangen wird. In Kapitel 3 wird ein grundlegendes Verständnis für den Begriff Cloud geschaffen. Das Kapitel mündet in der Betrachtung der Aspekte Sicherheit und Recht im Kontext von *Cloud Computing*. Kapitel 4 beschreibt die Modellierung sowie Implementierung eines Systems zur Ausbringung von Nutzungskontrolle und Provenance Tracking, welches in Kapitel 5 evaluiert und diskutiert wird. Den Abschluss der Arbeit bildet das Fazit und der Ausblick in Kapitel 6.

2. Grundlagen

Dieses Kapitel legt den aktuellen Stand zur Nutzungskontrolle sowie Provenance Tracking dar. Ferner wird eine selektierte Auswahl aktueller Forschungsprojekte zu diesen Themen aufgegriffen.

2.1. Nutzungskontrolle

Die Zugriffskontrolle auf Ressourcen, im Folgenden als Daten bezeichnet,[1] ist ein wesentliches Ziel in der Sicherheit von Informationstechnologien. Im Zuge dessen haben Forscher zahllose Modelle für Zugriffskontrolle vorgestellt, von denen sich nur die wenigsten letztendlich durchgesetzt haben. Die hierbei häufigst anzutreffenden – gemeinhin als „traditionellen Zugriffskontrollverfahren“ bezeichnet – sind Discretionary Access Controll (DAC), Mandatory Access Controll (MAC) und Role-based Access Controll (RBAC) [Mar+20]. Im Folgenden wird ein Szenario beschrieben, in denen diese jedoch unzureichend sind.

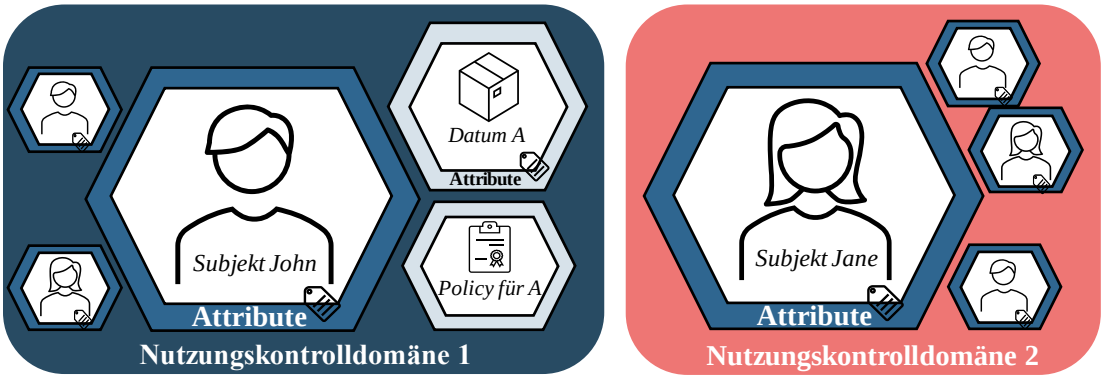

Abbildung 2.1.: Szenario zur Nutzungskontrolle

Man stelle sich vor, dass Subjekt John aus Abbildung 2.1 das Datum A, für welches bereits eine Policy definiert wurde, mit Subjekt Jane aus einer fremden Domäne (zum Beispiel einem eigenständigen Unternehmensnetzwerk) teilen möchte. Der Clou ist: Die Nutzung des Datums A

1 Im Kontext der Zugriffskontrolle spricht man i. d. R. von Objekten, aufgrund der Konsistenzhaltung mit dem Themengebiet Provenance wird im weiteren Verlauf das generische Wort Datum verwendet

darf dabei nur solange erfolgen, wie auch alle in der Policy für A genannten Bedingungen erfüllt sind. Sowohl die Autorisierung von Subjekten durch unterschiedliche Autoritäten, als auch die kontinuierliche Zugriffskontrolle von Daten, welche über die eigene Domänengrenze hinaus genutzt, transferiert und/oder kopiert werden können, ist durch „traditionelle Zugriffskontrolle“ nicht möglich. Die Abbildung 2.2 verdeutlicht noch einmal die möglichen Zustände eines Nutzungskontrollsystems, wobei die kontinuierliche Zugriffskontrolle durch den Zustand „Nutzung erlaubt“ erfolgt, welcher die Nutzungserlaubnis kontinuierlich neu bewertet.

Darüber hinaus sollen aktive und dynamische Entscheidungsprozesse ermöglicht werden, sodass diese nicht nur die Identität oder Attribute eines Subjekts, sondern auch kontextuelle Informationen, wie Standort oder Zeit berücksichtigen. Ferner soll es möglich sein, auch nach einer initialen Zugriffsfreigabe die Nutzungsrechte an einem Datum wieder entziehen zu können. Dies setzt voraus, dass sich der Urheber eines Datums sicher sein kann, dass gesetzte Policies auch über Domänengrenzen hinweg erzwungen werden können [ZSS08; Fan+16].

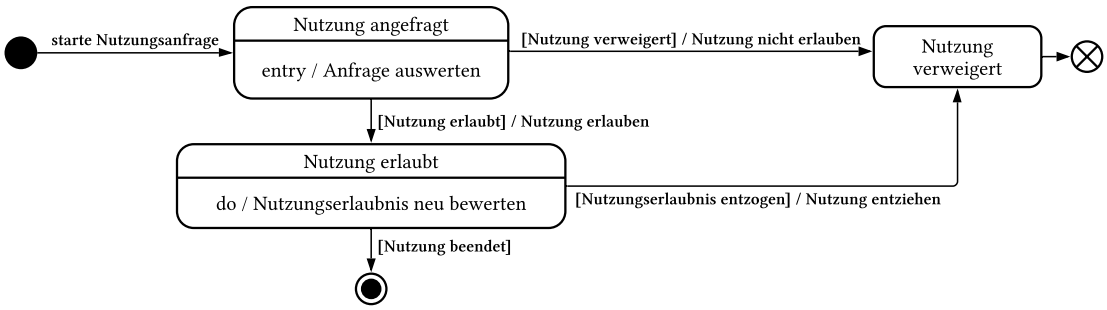

Abbildung 2.2.: Zustandsautomat zur Nutzungskontrolle

2.1.1. Theoretische Grundlage – UCON

Ein systematisch vereinheitlichtes Modell zur Nutzungskontrolle wurde bereits 2002 durch die Autoren Park und Sandhu mit der Bezeichnung $UCON_{ABC}$ in [PS02] formuliert. Darin baut Nutzungskontrolle auf den drei Konzepten Trust Management, Digital Rights Management (DRM) und Zugriffskontrolle auf. Jedes dieser Konzepte bringt Anforderungen mit sich, die ein Nutzungskontrollsystem erfüllen muss: So befasst sich Trust Management mit der Autorisierung von Personen, die einem bestehenden System unbekannt sein können [Wee01] und DRM mit der Kontrolle von Nutzung digitaler Informationen auf Seite des Nutzers [Bec03]. Außerdem ist Zugriffskontrolle als Prozess definiert, der Anfragen auf Objekte erlaubt oder verweigert [NDP17; PS02].

Das Modell ist eine Generalisierung der attributbasierten Zugriffskontrolle, engl. Attribute-based Acccess Control (ABAC). Es definiert daher auch Subjekte, die entweder direkt oder über Rechte attribuiert sind. Diese Rechte stellen Privilegien dar, die ein Subjekt an Objekten wahrnehmen kann. Ferner können Objekte original oder abgeleitet sein. Abgeleitet meint hierbei, dass ein Objekt durch eine Anfrage/Nutzung aus einem Originalobjekt erstellt wurde.

$UCON_{ABC}$ teilt sich in $UCON_A$, $UCON_B$, $UCON_C$ auf.[2] Die einzelnen Buchstaben stehen für: ***A**uthorization*, *O**b**ligation* und ***C**ondition*. Diese Kernelemente sind es, die feingranulare Entscheidungen abseits traditioneller Zugriffskontrolle ermöglichen:

Authorization Beantwortet die grundsätzliche Frage, ob einem anfragenden Subjekt die Nutzung eines Objekts gestattet ist.

Conditions Dies sind Bedingungen, die – bezogen auf kontextuelle Informationen – vor oder während der Nutzung eines Objekts erfüllt sein müssen. Als Beispiel hierfür lassen sich zeitliche Einschränkungen nennen, in denen der Zugriff nur zu bestimmten Zeiten gestattet ist.

Obligations Sind Bedingungen die der Anfragende vor oder während der Nutzung eines Objekts einhalten muss, um den Zugriff darauf aufrechtzuerhalten.

Der Entscheidungsprozess ist transaktionsbasiert, weshalb die Bearbeitungsreihenfolge von Anfragen für die zu treffende Entscheidung wichtig sein kann. So könnte man sich eine *Obligation* vorstellen die besagt: „Zugriff auf Objekt A sei nicht erlaubt, sofern Zugriff auf Objekt B bereits besteht“. Attribute sind vor oder während des Entscheidungsprozesses veränderbar. Das ist vor allem dann nützlich, wenn diese „konsumierbar“ sein sollen. Zum Beispiel könnte zur Begrenzung der gleichzeitigen Nutzung eines Objekts, dieses mit einem Zählerattribut versehen werden, welches die Anzahl der darauf zugreifenden Subjekte zählt [PS04].

Für Policysprachen, wie eXtensible Access Control Markup Language (XACML) von Organization for the Advancement of Structured Information Standards (OASIS), der Open Digital Rights Language (ODRL) und der Obligation Specification Language (OSL) existieren inzwischen Erweiterungen, die deren Syntax um die Kernelemente des $UCON_{ABC}$-Modells erweitert [Bie17; Hil+07; Col+10].

2.1.2. Verwandte Arbeiten

Die Forschung zum Thema Nutzungskontrolle lässt sich grob in folgende Bereiche einteilen: (i) Theoretische Grundlagen und Überlegungen, (ii) Spezifikation der Policies, (iii) Architektur von Nutzungskontrollsystemen, (iv) Etablierung gegenseitigen Vertrauens (Sicherstellung der

2 Man spricht auch von der „$UCON_{ABC}$ Familie“

Umsetzung von Policies). Während in den Anfängen der Nutzungskontrolle die Punkte (i) - (iii) im Mittelpunkt der Forschung standen, liegen nun die Schwerpunkte verstärkt auf (iii) - (iv) .

2.1.2.1. Etablierung gegenseitigen Vertrauens

Eines der größten Probleme in der Nutzungskontrolle ist die Etablierung des gegenseitigen Vertrauens im Kontext des *Ubiquitous Computing*. Die hierfür notwendigen Sicherheitsanforderungen wurden 2008 von Zhang, Seifert und Sandhu in [ZSS08] aufgestellt. Die Autoren stellen darin die Notwendigkeit eines *Trusted Subsystem* dar, welches in der Lage sein muss – für den Datenurheber verifizierbar – die Policies auf der Zielplattform zu erzwingen. Sie bemängeln darin, dass vorangegangene Forschung sich vorrangig auf die konzeptionelle Architektur von Nutzungskontrollsystemen sowie der Spezifikation von Policysprachen bezog. Die eigentlichen Mechanismen zur Umsetzung der Nutzungskontrolle wurden durch DRM und kryptografische Algorithmen realisiert. Nach Ansicht der Autoren sind diese jedoch nicht in der Lage, den oben geforderten aktiven, dynamischen Entscheidungsprozess abzubilden. Vor allem sei es damit nicht möglich, Garantien zur Umsetzung von definierten Policies im *Ubiquitous Computing* zu geben. Sie schlussfolgern daraus, dass die Kontrolle über Ressourcen und Sicherheitsmechanismen eines Geräts über ein *Trusted Subsystem* geregelt sein muss. Dieses muss auf Betriebssystemebene arbeiten, um eine Ende-zu-Ende-Sicherheit garantieren zu können, in der das *Trusted Subsystem* nicht umgangen werden kann. Auf diese Weise soll Integrität, Vertraulichkeit, Datenschutz und Verifizierbarkeit durch den Datenurheber sichergestellt werden. Im Rahmen ihrer Arbeit gehen sie weiter auf die von ihnen entwickelte *Policy Enforcement Architecture* ein, die mittels Trusted Platform Modules (TPMs)[3] und Integritätsmessungen/-verifikation sowie SELinux eine Sicherheitskette von Hardwareebene bis zur Laufzeitebene aufspannt, um ein solches *Trusted Subsystem* zu realisieren. Die konkrete Umsetzung soll an dieser Stelle aber nicht weiter vertieft werden.

2.1.2.2. Steuerung des Datenflusses in der Nutzungskontrolle – LUCON

LUCON ist ein Policy-Framework zur Forcierung bestimmter Datenflüsse in nachrichtenbasierten Systemen. *LUCON* erhebt damit den Anspruch, dass $UCON_{ABC}$ Modell auf einer niedrigeren Abstraktionsebene um eine Steuerung des Datenflusses auf Grundlage von Policies zu erweitern. Die Notwendigkeit eines solchen Frameworks besteht darin, dass Daten von einer Quelle, zum Beispiel einem Sensor, nicht direkt zu einer Senke fließen sollen. Unaufbereitete Daten können Informationen durchsickern lassen, die Rückschlüsse auf Verfahren oder Personen zulassen.

3 `https://trustedcomputinggroup.org/resource/tpm-library-specification` (besucht am 06. 11. 2020)

Primär für Internet of Things (IoT)-Systeme vorgesehen, besteht ein System, in dem *LUCON* zum Einsatz kommt, aus mindestens einer *Trusted Domain*. In einer solchen Domäne sind die Routen von Nachrichten nicht öffentlich. Die Nachrichten (z. B. Sensordaten) werden mit Labels versehen und anhand entsprechender Policies geroutet. *LUCON* selbst ist eine Domain Specific Language (DSL), mit der Regeln für Routen definiert werden. Um das Framework in bereits bestehende Systeme zu integrieren, braucht es mindestens einen Integrationspunkt und *Services*, die transparent mit den Labels umgehen können.

Die vorliegende Arbeit wird *LUCON* nicht verwenden, da sich die Untersuchung auf die Ausbringung einer Nutzungskontrollinfrastruktur beschränkt. *LUCON* liegt jedoch eine Abstraktionsebene höher und baut auf einer bereits bestehenden Nutzungskontrollinfrastruktur auf. Dadurch dass es auf einer höheren Abstraktionsebene ansetzt als Netzwerklösungen, wie Multiprotocol Label Switching (MPLS), wird den Nutzern mehr Flexibilität und ein mächtiges Werkzeug für das Routing von Nachrichten gegeben. Die Flexibilität hat jedoch zusätzliche Verzögerungen zur Folge, da die *LUCON* Policies beim Routing ausgewertet werden müssen [SB18].[4] Ferner sollte untersucht werden, ob *LUCON* in bereits vorhandene Routinglösungen, wie Software-defined Networking (SDN), eingepflegt werden kann. Die Autoren Kern und Anderl zeigten hierzu in [KA20], dass Nutzungskontrolle im Zusammenspiel mit SDN durchaus möglich ist.

2.1.2.3. Weitere Arbeiten

Einen andersartigen Ansatz stellt *MUCON* dar. *MUCON* steht für $multi-UCON$ und versucht DRM und Verschlüsselungstechnologien in einem Protokoll zu vereinen, um die Nutzungskontrolle umzusetzen. Nach Meinung der Autoren soll das *MUCON*-Protokoll besonders effektiv, sicher, zuverlässig und einfach für den Einsatz in der Cloud zu implementieren sein. Da *MUCON* jedoch stark auf DRM Technologie aufbaut und die Verarbeitung von Daten innerhalb der Cloud stattfindet, wird es im Folgenden nicht weiter betrachtet [Fan+16].

Ähnlich verhält es sich mit [Ali+10], in der die Autoren Nutzungskontrolle als Software as a Service (SaaS)-Lösung etablieren wollen. Die Vorstellung der Autoren ist, dass Nutzer über den Browser mit einer als SaaS ausgerollten Nutzungskontrolle interagieren können.[5] Dieses Vorgehen ermöglicht aus Sicht des Cloud Service Provider (CSP) mehr Kontrolle über die Daten. Für den Datenurheber ist dies nicht der Fall, denn er übergibt die Kontrolle an den Provider. Nichtsdestotrotz sind die darin aufgestellten Forderungen nach Effizienz, Benutzerfreundlichkeit, einer mächtigen Policysprache und Standardisierung Punkte, die vor allem für

4 Damit sind Anzahl Policies und Domänengröße ein nicht zu vernachlässigender Skalierungsfaktor
5 Vergleichbar mit Produkten wie Outlook, G Suite, Slack, etc.

den operativen Betrieb der Nutzungskontrolle wichtig sind.

Davon abzugrenzen sind Anwendungsfälle, in denen es um Nutzungskontrolle bezüglich Cloud Ressourcen geht. Hierzu stellen die Autoren in [Laz+12; Laz+16] ein enstprechendes Framework für CSPs vor. Die Autoren in [Ana+14] betrachten zusätzlich den Fall des Zusammenschlusses von mehreren CSPs.

Im Kontext der Industrie 4.0 und IoT ist eine Erweiterung der OPC Unified Architecture (OPC-UA) durch Nutzungskontrolle, wie sie in [Mar+20] vorgestellt wird, interessant. Open Platform Communications (OPC) ist ein Kommunikationsprotokoll, das für Systeme im industriellen Umfeld von der eigens dafür gegründeten OPC-Foundation standardisiert wurde, um sicheren, zuverlässigen, gleichartigen und herstellerunabhängigen Datenaustausch zu ermöglichen.[6]

Speziell im Kontext von Smart Home IoT lässt sich das *UCIoT* Modell anführen, das auf einer Peer-to-Peer (P2P) Architektur aufbaut [Mar+17]. Der Zugriff auf Attribute und das Session Management für die kontinuierliche Zugriffskontrolle geschieht über die verteilte Datenbank Cassandra.[7]

2.2. Provenance Tracking

Das Wort Provenance stammt vom französischen Wort *provenir* und bedeutet so viel wie „herkommen/herauskommen von“. In Bezug auf die Provenance eines Datums[8] lassen sich zwei grundsätzliche Fragestellungen ableiten: Die Bestimmung dessen Herkunft sowie die Aufzeichnung darüber, was mit dem Datum im Verlauf der Zeit passierte. Die Aufzeichnung ist also ein Resultat von Zustandsveränderungen an dem ursprünglichen Datum, ähnlich einem Stammbaum. Allem voran ist Provenance ein Begriff, der durch die Kunst geprägt wurde. Ist die Provenance eines Gemäldes nicht nachvollziehbar, so wird dieses erst einmal mit Skepsis betrachtet. Das Gemälde könnte eine Fälschung, manipuliert oder gar gestohlen sein. Die Echtheit eines Gemäldes kann erst dann nachvollzogen werden, wenn Verständnis darüber besteht, wie dessen aktueller Zustand zustandekam [Mor+08].

Die Möglichkeiten, die sich durch das Erfassen von Provenance-Daten ergeben, sind für eine Vielzahl von informationsverarbeitenden Systemen in den unterschiedlichsten Industrien relevant. So können hierdurch datengetriebene Prozesse analysiert und die Einhaltung von zu erfüllenden Regeln überprüft werden. Im Finanz- und Medizinsektor ergibt sich sogar eine

6 Am 25.06.2020 wurde Google Cloud Mitglied der OPC Foundation und kündigte an, OPC-UA als Integrationspunkt für ihre Data Analytics und AI Dienste zu verwenden [OPC20]

7 `https://cassandra.apache.org` (besucht am 02. 09. 2020)

8 In der Literatur wird hierfür oft der Begriff „Dokument“ verwendet

gesetzliche Notwendigkeit der Erfassung von Provenance-Daten [MF06]. Aber auch im Rahmen des Art. 13 und 15 Datenschutz-Grundverordnung (DSGVO) beschreibt Bier in [Bie17] die Erfassung der Provenance von Nutzerdaten als notwendigen Baustein zur Realisierung der Betroffenheitsrechte. Daten- und Eventlogs alleine könnten dies ohne vollständiges a priori Wissen über das Geschäftsmodell nicht leisten [Cur+08]. Allerdings kann der Detailgrad einer Provenance stark variieren. Der Grund hierfür liegt in der Art und Weise wie das Provenance Tracking realisiert wird und auf welcher Abstraktionsebene die Erhebung stattfindet. So werden sich Provenance-Daten, die auf Betriebssystemebene gesammelt wurden, von Daten, die auf Ebene der Business-Logik eines *Services* gesammelt wurden, unterscheiden. Erschwerend kommt hinzu, dass die meisten Systeme gar nicht dafür ausgelegt sind, diese zu sammeln. Ferner setzen immer mehr Systeme auf einer verteilten, offenen Architektur auf, die sich dynamisch zusammensetzen kann. Probleme, die dabei in der Forschungsliteratur genannt werden, sind: (i) Technologische Unabhängigkeit, (ii) ein gemeinsames Datenmodell, (iii) Sammeln und Auswerten von Provenance-Daten, (iv) Skalierbarkeit und (v) Datensicherheit (mit den Eigenschaften: *Confidentiality, Privacy, Integrity, Availability, Unforgeability*) [Mor+08; Zaf+17].[9]

2.2.1. Eine Taxonomie für Provenance

Die Autoren Simmhan, Plale und Gannon stellen in [SPG05] eine Taxonomie (siehe hierzu Abbildung 2.3) zum Themengebiet Provenance vor.[10, 11] Da diese einen konzeptionellen Überblick über das Themengebiet gibt, wird sie im Folgenden wiedergegeben.

2.2.1.1. Anwendungsgebiete

Die Taxonomie teilt die Anwendendung von Provenance in fünf Kategorien auf. Hier sei jedoch angemerkt, dass in konkreten Anwendungszenarien die Grenzen einer solchen Kategorisierung vermutlich verschwimmen.

Data Quality Anwendungen, in denen Aussagen über die Qualität und Zuverlässigkeit von Daten zu treffen sind.

9 An dieser Stelle sei auf [HSW07] zum Thema *Secure Provenance* verwiesen

10 Die Darstellung wurde dahingehend modifiziert, als dass Unterknoten zu *Representation Scheme* nicht aufgeführt sind und *Syntactic and Semantic Information* sowie *Process and Data Oriented* als jeweils ein Knoten zusammengeführt wurden

11 Diese Taxonomie wurde aufgrund ihrer Anschaulichkeit ausgewählt. Die Taxonomie in [CCM09] bezieht sich speziell auf die Provenance von wissenschaftlichen Daten und die Autoren in [Zaf+17] legen verstärkt Fokus auf das Thema Sicherheit

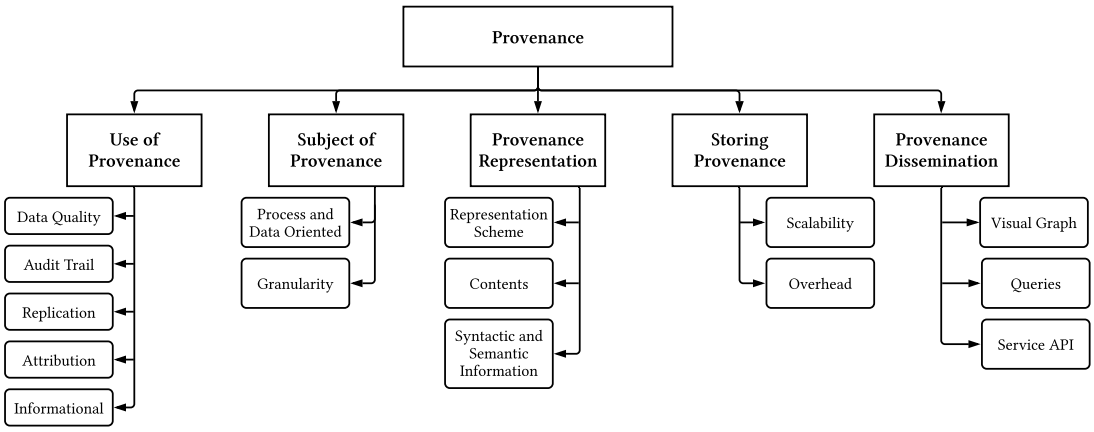

Abbildung 2.3.: Provenance Taxonomie in [SPG05]

Audit Trail Zur Überwachung und Dokumenation des Datenflusses. Effektiv sollen hiermit Prozesse, Aktivitäten und Ergebnisse nachvollziehbar gemacht werden. Im medizinischen Bereich muss zum Beispiel nach dem Health Care Portability and Accountability Act (HIPAA) von 1996 jeder Zugriff und Veränderung von medizinischen Daten aufgezeichnet werden [HSW07].

Replication Recipes Wiederholbarkeit der Erzeugung und Transformation von Daten auf Grundlage aufgezeichneter Provenance-Daten.

Attribution Ermittlung des Datenurhebers.

Informational Erlaube möglichst generische Suchanfragen auf den Provenance-Daten (zum Beispiel auf Basis von Metadaten), um diese zu erkunden oder zusätzlichen Kontext für deren Interpretation zu schaffen.

2.2.1.2. Erhebung

Eine Provenance kann auf direktem (*Data Oriented*) oder indirektem Weg (*Process Oriented*) erhoben werden.[12] Der direkte Weg hat das explizite Tracking der Provenance eines Datums zur Folge, wohingegen im indirekten Fall anhand der stattfindenden Transformationsprozesse und deren Ein- und Ausgabe die Provenance eines Datums abgeleitet wird.

Ferner ist die Granularität mit der die Provenance erfasst wird von Bedeutung. Je feingranularer die Erhebung vorgenommen wird, umso mehr Informationsgehalt hat diese.

12 In [CCM09] auch als *eager and lazy* bezeichnet

2.2.1.3. Repräsentation

Die Repräsentation einer Provenance lässt sich – wie bereits die Erhebung – auf direktem oder indirektem Weg realisieren. Der direkte Weg ist die Repräsentation mithilfe von *annotations*. Hierbei handelt es sich um Metadaten über Transformationsprozesse, die auf ein Datum wirkten. Die gesammelten Metadaten bilden in ihrer Gesamtheit dann die Provenance.

Der indirekte Weg sieht vor, dass die auf ein Datum gewirkte Transformation invertierbar ist, daher kommt auch deren Bezeichnung *inversion method*. Für die Ableitung einer Provenance reicht die Ausgabe und das Wissen über die gewirkte Transformation aus, um auf das ursprüngliche Datum zu schließen.

Im Zuge des *International Provenance and Annotation Workshop* im März 2006 und dem darauffolgenden in 2007, wurde in einer gemeinsamen Anstrengung der Teilnehmer ein Datenmodell für Provenance mit dem Namen *Open Provenance Model* entwickelt [Mor+11]. Für die Anwendung von Provenance im Web hat das World Wide Web Consortium (W3C) den PROV Standard erarbeitet.[13] Dieser beeinhaltet unter anderem ein Datenmodell und Spezifikationen zur Serialisierung von Provenance-Daten. Beide Ansätze sind sich aufgrund gemeinsamer Autoren sehr ähnlich.

2.2.1.4. Speicherung

Ein wichtiger Punkt für die Skalierbarkeit eines Provenance-Systems ist die Speicherung. Je nach Detailgrad können die anfallenden Provenance-Daten die Speichergröße des zu trackenden Datums übersteigen. Im Falle der Nutzung von *annotations* kann es von Vorteil sein, immer nur den vorangegangenen Transformationsprozess (der damit das Datum erstellt hat), zu speichern, um dann bei Bedarf die Historie des Datums rekursiv anhand seiner Vorfahren ableiten zu können.

Der Speicherort der Provenance kann auf demselben oder auf einem anderen Speichersystem wie das Datum liegen. Dies schließt auch das Speichern der Provenance in dem Datum selbst mit ein. Provenance-Daten, die nach einer gewissen Zeit nicht abgerufen wurden, können gegebenfalls archiviert werden und so Platz für neue schaffen.

2.2.1.5. Schnittstelle zum Nutzer

Mit der Kategorie *Provenance Dissemination* ist die Nutzung der Provenance-Daten aus Sicht des Nutzers gemeint. Konkret geht es darum, wie Nutzer Anfragen darauf ausführen können, wie generisch diese sein dürfen und wie verwertbar die Ausgabe ist. Als Beispiel stelle man sich

13 `https://www.w3.org/TR/prov-overview` (besucht am 19. 09. 2020)

eine *Service API* vor, die es Nutzern erlaubt, nach bestimmten Werten in den Metadaten der Provenance zu suchen. Durch die Möglichkeit einer solchen Durchsuchung, können zusätzliche semantische Informationen gewonnen und für Interpretationen genutzt werden.

2.2.2. Provenance Tracking im Kontext von Cloud Computing

Die Forschung zum Themengebiet Provenance konzentrierte sich zunächst im Bereich von Datenbanksystemen. Von dort aus breitete sie sich auf Dateisysteme und Workflowsysteme (oft mit Schwerpunkt e-Science) aus. Die neuere Forschung hat Provenance im Bereich von verteilten Systemen, wie Cloud Computing, als Schwerpunkt.

2.2.2.1. Einschränkungen

Nach den Autoren Sakka, Defude und Tellez in [SDT10] lässt sich Provenance in drei Verantwortlichkeitsbereiche einordnen:

Rechtlich Bezogen auf regulatorische Anforderungen und gesetzliche Bestimmungen. Durch die verteilte Natur der Cloud treten territoriale Probleme auf, nicht immer ist klar, wer die rechtliche Autorität ist und in der Folge, welche Gesetze anzuwenden sind.

Geschäftlich Systeme sind immer in einen Kontext eingebettet. Die Art und Weise, in der das Provenance Tracking erfolgt, hängt von der konkreten Anwendungsdomäne ab.

Technisch Hier treten Probleme der Interoperabilität, Aspekte der Sicherheit und des Vertrauens sowie Skalierbarkeit auf. Zum Beispiel ist es sicherheitsrelevant, dass einem Administrator die Einsicht in die Provenance-Daten eines Kunden nicht erlaubt sein sollte, gleichzeitig sollte der Kunde durch diese Möglichkeit nicht zu viel über Systeminterna erfahren dürfen.

Ferner stellen sie in ihrem Artikel konkrete Herausforderungen an Provenance in der Cloud vor, die nachfolgend thematisiert werden.

2.2.2.2. Herausforderungen

Im Kontext von Big Data nennen die Autoren in [Wan+15] vier konkrete Herausforderungen: (i) Deren Größe, (ii) den zusätzlichen Aufwand beim Sammeln, (iii) die Schwierigkeit der Speicherung und Integration verteilter Provenance-Daten in Verbindung mit den Lebensphasen diesbezüglicher Daten und (iv) das Erfassen von Informationen über die Laufzeitumgebung. Mitunter ist bereits die Identifizierung von Daten, die sich zwischen unterschiedlichen CSP bewegen, problematisch. Unter der Annahme, das die Datenidentifikation verschiedener CSP auch unterschiedlich sein kann, könnte sich in der Folge die technische Repräsentation einer

Provenance unterscheiden und so eine korrekte Identifizierung verhindern. Hierfür reichen Identifikationsverfahren, wie das Nutzen von Hashwerten oder Unique Identifiers (UIDs) alleine nicht aus [SDT10].

Die Autoren Bates et al. machen in [Bat+13] auf das Problem aufmerksam, dass Daten transparent repliziert und an mehreren Orten gespeichert werden können. Als Beispiel nennen sie hierfür Regulierungen, die es verbieten, in den USA angefallene Daten in der EU zu speichern.[14] Deshalb entwickelten sie den Prototyp eines *Cloud Provenance Authority Systems*, das die Aufgabe hat, sowohl den Zugriff als auch die Erstellung von Provenance-Daten zu schützen und zu managen. Die Schutzwürdigkeit dieser Daten ist nicht nur aus Integritätsgründen gegeben, sondern sie zeigt sich auch in Szenarien, in denen sensible Informationen durch die Provenance gewonnen werden können.

Ferner könne die Cloud per se nicht als vertrauenswürdiger Bereich für Daten angesehen werden. Infolgedessen ist eine Manipulation und Fälschung von Provenance-Daten möglich. Um dem entgegenzuwirken müssen folgende Eigenschaften gegeben sein: (1) Schutz der Integrität von Provenance-Daten, das heißt mögliche Manipulations- oder Fälschungsversuche werden erkannt. (2) Verfügbarkeit, wodurch die Integrität und Korrektheit der Provenance-Daten jederzeit überprüfbar ist. (3) Vertraulichkeit, die Provenance-Daten sollten einer Zugriffskontrolle unterliegen. (4) Atomarität bezüglich der Veränderung von Provenance und Datum sowie (5) Konsistenzhaltung von Datum und Provenance [SDT10].

Die Autoren sehen drei grundlegenden Architekturen eines Provenance-Systems vor: Zentral, hybrid (mit import/export) und föderiert. Der zentrale Ansatz nutzt eine eindeutige vertrauenswürdige Autorität, die Provenance als Service anbietet. Im hybriden Ansatz können Provenance-Daten zwischen einer zentralen Instanz und weiteren externen Clouds importiert oder exportiert werden. Im föderierten Fall agiert jede Cloud autonom. Im Kontrast dazu stellen die Autoren in [MMS10] eine native Unterstützung zur Erfassung von Provenance-Daten seitens der CSPs in Aussicht. Durch eine solche Umsetzung könnten sowohl diese als auch deren Kunden profitieren. Jedoch müsste hierfür ein gemeinsamer Standard oder zumindest ein gemeinsames Application Programming Interface (API) definiert werden, um einen *Vendor Lock-in* zu vermeiden.

2.2.2.3. Weitere Arbeiten

In [She+16; Bie17; MS10] stellen die Autoren vor, wie sich Provenance und Zugriffskontrollverfahren gegenseitig ergänzen können. So ist die Möglichkeit, Nutzer mit einer Provenance in Verbindung zu bringen, für eine feingranularere und vertrauensvollere Provenance interessant.

14 Dieses Phänomen wird auch als *Geolocating* bezeichnet. Im umgekehrten Fall greift die DSGVO

Umgekehrt lässt sich auf Basis von Provenance-Daten Zugriffskontrollentscheidungen treffen. Die Autoren Park, Nguyen und Sandhu versuchten hierzu in [PNS12] die Theorie für eine *Provenance-based Access Control* aufzustellen.

Der in [Sue+13] bezeichnete S2Logger ist nach Angaben der Autoren die erste Software, die umfangreich die Datenaktivität in den Knotenpunkten einer Cloud-Umgebung aufzeichnet. Der S2Logger arbeitet auf Kernelebene und sammelt lokal die Provenance-Daten der auf einem Knoten laufenden virtuellen Maschinen ein und reicht diese dann an eine zentrale Datenbank weiter. Das Aufzeichnen der Daten in den virtuellen Maschinen geschieht wahlweise über *System Events* oder die *Linux Security Module API*, wobei seit Linux Kernel 2.6 die Aufzeichnung über *System Events* nur über komplexe Umwege möglich sei. Die Autoren Ko und Will [KW14] wollen eine bessere Alternative zu S2Logger entwickelt haben. Ihr Tool soll darüber hinaus das Zeitsynchronisationsproblem mit mehreren Maschinen behoben haben, ein effizienteres Wachstum der Log-Größe und manipulationssichere Log-Einträge vorweisen können.

Provenance Tracking von Sensordaten auf Netzwerkebene wird von den Autoren in [ZSS08] diskutiert. Hierbei werden Provenance-Daten im *Interpacket Delay* übertragen. Diese Methode wird auch als *Watermarking* bezeichnet.

Für das Teilen von Patientendaten über unterschiedliche CSP entwickelten die Autoren in [Xia+17] eine auf Blockchain basierende Technologie namens *MeDShare*. Mithilfe von *Smart Contracts* und Zugriffskontrolle soll die Provenance von Daten zurückverfolgt und Zugriffe auf Patientendaten widerrufen werden können. Ähnlichen Fokus legen die Autoren in [Lia+17] mit *ProvChain*. Dieses soll vor allem weitreichende Sicherheitseigenschaften, wie eine manipulationssichere Provenance, Datenschutz, Zuverlässigkeit sowie geringen Overhad für Cloud Storage-Anwendungen bieten.

2.3. Forschungsprojekte

Dieser Abschnitt widmet sich ausgesuchten Forschungsprojekten, die sich mit der Handhabung von Daten durch andere beschäftigen. Des Weiteren erfolgt eine Einführung zur Referenzarchitektur der Nutzungskontrolle und des Provenance Trackings, die im weiteren Verlauf der Arbeit wichtig sein wird.

2.3.1. Cloud Accountability Project – A4Cloud

Das von der EU geförderte Cloud Accountability Project (A4Cloud)[15] hat das Ziel, die Aspekte der Verantwortlichkeiten von CSPs hinsichtlich personenbezogener Daten auf einer konzep-

15 `http://cloudaccountability.eu` (besucht am 09. 10. 2020)

tionellen Ebene zu definieren und Herausforderungen, die sich durch die Cloud ergeben, aufzugreifen. Neben Richtlininien, Empfehlungen und Metriken wurde ein mehrschichtiges Modell für Verantwortlichkeit geschaffen. Die funktionalen Elemente des daraus entstandenen Modells – die Mechanismen – formen wiederum ein Framework.[16, 17] Ferner wurden im Rahmen des Projektes prototypische Tools für CSPs entwickelt, welche den Nutzern mehr Kontrolle über deren Daten sowie mehr Transparenz im Hinblick auf deren Verarbeitung in der Cloud geben sollen. Dies umfasst unter anderem ein *Policy Configuration and Enforcement System* sowie ein *Monitoring Tool* zur konkreten Überprüfung durch den Nutzer.

Die Autoren sehen die Herausforderung vor allem im interdisziplinären Charakter und in der Integration verschiedenster Sichtweisen auf die Problemstellung, welche rechtlicher, ethischer, wirtschaftlich-sozialer und technischer Natur sind. Zwar stellt die Referenzarchitektur eine Sammlung an Funktionen zur Schaffung von Verantwortlichkeit dar, die eigentliche Arbeit bestand jedoch darin, den Begriff Verantwortlichkeit zu erfassen und daraus Anforderungen zur Schaffung dieser abzuleiten [A4Cloud].

2.3.2. International Data Spaces (IDS)

Der IDS entstand aus einem Ende 2014 begonnen Forschungsprojekt der Fraunhofer-Gesellschaft, das die Entwicklung einer Referenzarchitektur zur Schaffung virtueller Datenräume vorsieht. In diesen Räumen sollen zertifizierte Teilnehmer sicheren Datenaustausch praktizieren und gleichzeitig die Souveränität über ihre Daten behalten können. Nutzer können die virtuellen Räume über Konnektoren betreten, die als *Gateway* fungieren (siehe Abbildung 2.4) [GS20; FR-AISEC]. Je nach Art des Konnektors sind unterschiedliche Sicherheitseigenschaften gegeben, so bieten zum Beispiel Konnektoren mit dem Sicherheitsprofil „trust“ die Möglichkeit einer gegenseitigen Verifikation der Integrität und nicht nachahmbare Identitäten an [DIN-SPEC-27070]. Der App-Store ermöglicht die Ausführung von Anwendungen innerhalb des Konnektors und *Broker* stellen Dienste dar, die das Verwalten und Durchsuchen von Metadaten erlauben, welche durch *Data Provider* bereitgestellt wurden. Nicht in Abbildung 2.4 zu sehen sind das *Clearing House* sowie *Identity Provider* und *Vocabulary Provider*. Das *Clearing House* dient zur Buchführung und Abrechnung des Datenaustauschs. *Identity Provider* stellen einen Dienst zur Verwaltung, Validierung und Überwachung von Identitäten im IDS dar. Um syntaktische sowie semantische Interoperabilität zwischen Daten zu gewährleisten, existieren *Vocabulary Provider*, die Beschreibungen von Daten vorhalten. Sollten Teilnehmer

16 Der Entwurf einer Referenzarchitektur ist hier zu finden: `http://cloudaccountability.eu/content/reference-architecture` (besucht am 09. 10. 2020)

17 Für die einzelnen Funktionen wurde eine Sammlung an Tools in Form eines Toolkits entworfen: `http://cloudaccountability.eu/content/a4cloud-toolkit` (besucht am 09. 10. 2020)

nicht die notwendige Infrastruktur zur Teilnahme am IDS haben, so können *Service Provider* genutzt werden, welche dann stellvertretend für diese auftreten. *Service Provider* können auch insofern attraktiv für Nutzer sein, als dass sie für diese zusätzliche Dienste, wie zum Beispiel *Data Analytics*, bereitstellen können [IDS-RAM, S.23-25].[18, 19]

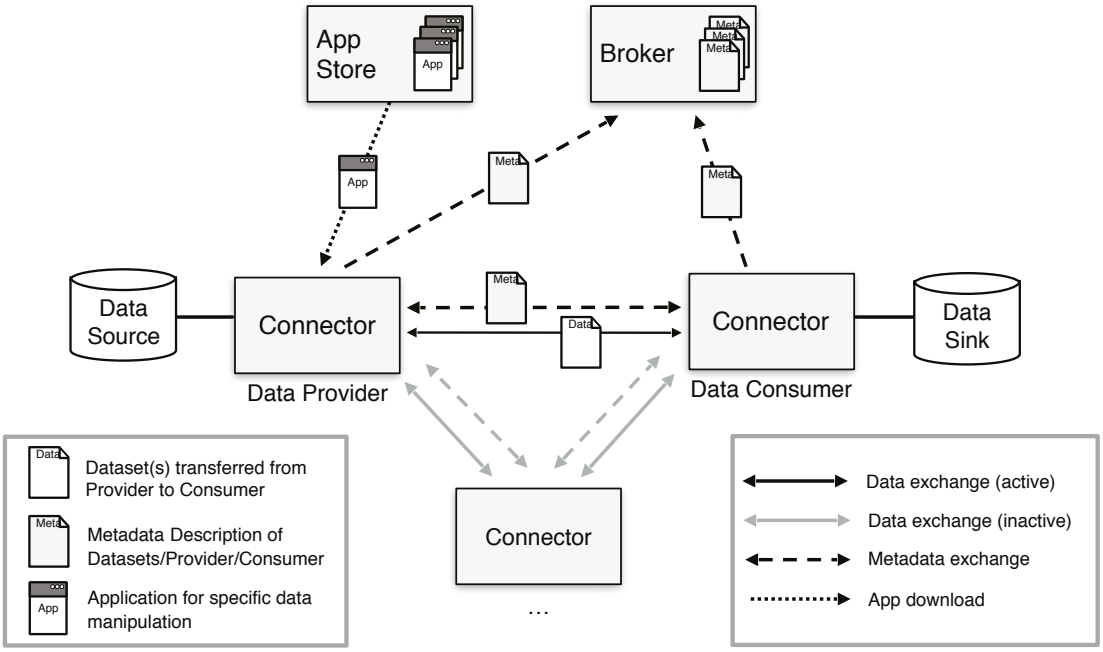

Abbildung 2.4.: Interaktion der wichtigsten Komponenten im IDS [IDS-RAM, S.60]

Seit 2016 steht der IDS unter Leitung der International Data Spaces Association (IDSA), einem gemeinnützigen Verein, der die Arbeiten seiner Mitglieder bündeln soll. Zu diesen Arbeiten gehören: Der Entwurf einer Referenzarchitektur, Bemühungen zur Standardisierung des Datenaustausches sowie Referenzimplementierungen hinsichtlich der einzelnen Komponenten des IDS [IDS-FS].

2.3.2.1. Nutzungskontrolle im IDS

Neben der Möglichkeit, den Zugriff auf Dienste und Daten mit Zugriffskontrolle zu versehen, bietet der IDS Nutzungskontrolle an, welche bereits im Abschnitt 2.1 besprochen wurde. Hierfür wird bei der Initialisierung eines Datenaustausches eine entsprechende Policy – oder zumindest eine Referenz darauf – angehängt. Der IDS sieht in der weiteren Verarbeitung dieser Daten

18 Als Beispiel sei der *Data Intelligence Hub* der Telekom genannt `https://dih.telekom.net` (besucht am 22. 10. 2020)

19 Nutzer die hierbei als *Data Provider* auftreten wollen, müssen Daten zunächst zum *Service Provider* hochladen

Mechanismen vor, die anhand der angehängten Information eine kontinuierliche Überwachung und Kontrolle zulassen. Dies setzt allerdings eine Umgebung und darin befindliche Systeme voraus, in denen gesetzte Policies auch forciert werden. Möglichkeiten zur Umsetzung und Schaffung dieses Vertrauens können in Form von Zertifizierungen, organisationsbezogenen Regeln oder rechtlich bindenden Verträgen, bis hin zu technischen Mechanismen und Systemen reichen [IDS-RAM].

Der Autor der vorliegenden Arbeit ist der Meinung, dass nur durch eine zielführende Abstimmung der genannten Möglichkeiten nutzbare Systeme enstehen. Die Referenzarchitektur des IDS hat sich dahingehend sehr stark auf die technischen Möglichkeiten fokussiert und ausgerichtet. Die Referenzarchitektur zur Nutzungskontrolle im IDS ist in Abbildung 2.5 zu sehen.[20]

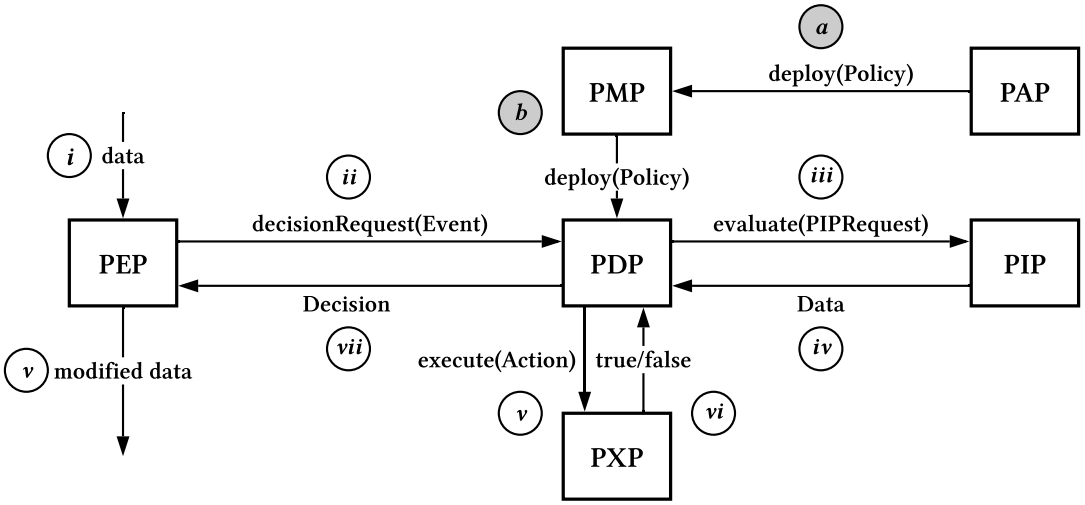

Abbildung 2.5.: Referenzarchitektur zur Nutzungskontrolle im IDS [IDS-UC]

Die Abbildung beschreibt den Datenfluss, welcher durch ein unter Nutzungskontrolle stehendes Datum angestoßen wird. Der Policy Enforcement Point (PEP) läuft auf demselben System, in dem sich das Datum befindet – entweder als importierte Bibliothek in Anwendungen oder als im Hintergrund laufender Prozess (*Daemon*).[21] Der PEP überwacht und kontrolliert den Zugriff auf das Datum, um die Nutzungskontrolle umzusetzen. Entscheidungen zur konkreten Nutzung eines Datums werden jedoch vom Policy Decision Point (PDP) getroffen. Dieser bekommt vom PEP Events zugeschickt, anhand derer eine Neubewertung der Nutzungskontrollentscheidung vorgenommen wird. Sollte der PDP hierbei zusätzliche Informationen benötigen, wie

20 Anzumerken ist hierbei die große Ähnlichkeit zum Datenfluss der XACML-Spezifikation
21 Der Hintergrundprozess kann als *Kernel Level* oder *User Level* Prozess laufen

zum Beispiel Attributwerte eines Datums, so können Policy Information Points (PIPs) befragt werden. Des Weiteren können durch den Policy Execution Point (PXP) Aktionen ausgeführt werden, wobei die Ausgabe den Erfolg oder Misserfolg der Aktion angibt.[22] Die Entscheidung wird letztendlich dem PEP mitgeteilt, der diese auf dem System mit dem Datum forciert. Der durch `(a,b)` dargestellte Ablauf ist nur dann notwendig, wenn Policies unabhängig von Daten gespeichert werden können.[23] Über den Policy Administration Point (PAP) können Policies spezifiziert und vom Policy Management Point (PMP) an PDPs verteilt werden. Weitere Aufgaben des PMP betreffen die Verwaltung und das Widerrufen von Policies [IDS-RAM].[24]

Der Autor Bier schränkt in [Bie17, S.108] die Architektur weiter ein, indem die Komponenten jeweils lokalen Domänen zugeordnet sind und sich bei dem für die Domäne zuständigen PMP registrieren müssen. Durch Verbinden dieser PMPs ensteht ein hierarchisches Netzwerk, wodurch eine gezielte und lokale Anwendung von Nutzungskontrolle möglich ist. Bier vergleicht diesen Ansatz mit dem Domain Name System (DNS), welches einen Dienst zur Namensauflösung von Domainnamen bereitstellt. Diese Architektur sorgt jedoch zum einen dafür, dass man sich mit möglichen Konsistenzproblemen zwischen PMPs beschäftigen muss[25] und zum anderen erhöht sich der administrative sowie operative Aufwand. Nach dem *CAP-Theorem* kann ein verteiltes System niemals alle drei Eigenschaften ***C**onsistency, **A**vailability and **P**artition Tolerance* gleichzeitig erfüllen [Bre00]. DNS-Systeme erfüllen die Punkte *AP*, im Falle der Nutzungskontrolle stellt sich die Frage, ob Systeme mit stärkerem Fokus auf *Consistency* erstrebenswerter sind.

PEP, PIP und PXP können prinzipiell auf demselben System ausgeführt werden, auf dem sich das Datum befindet. So kann der PIP Informationen über das System zur Verfügung stellen und der PXP Aktionen, wie das Löschen von Daten, auf diesem ausführen.

2.3.2.2. Provenance Tracking im IDS

Provenance Tracking im IDS hat den Zweck, Transparenz zu ermöglichen und Verantwortlichkeiten für eine spätere Überprüfung festzuhalten. Neben dem Erzwingen von Nutzungskontrolle können PEPs auch zum Erfassen der Provenance genutzt werden. Diese müssen lediglich als passiver Beobachter der Nutzung und des Datenflusses auftreten. Das beobachtete Verhalten wird daraufhin semantisch interpretierbar repräsentiert und aggregiert. Dies geschieht in Form von Events, die an einen Provenance Storage Point (ProSP) geschickt werden. Der Provenance

22 Die Einschränkung, dass lediglich der Erfolg oder Misserfolg ausgegeben wird, ist in [IDS-UC] nicht näher begründet. Dem Autor der vorliegenden Arbeit erscheint die Aufhebung dieser Einschränkung jedoch sinnvoll, um zum Beispiel das Komponieren von Aktionen zu ermöglichen

23 Im anderen Fall spricht man von *Sticky Policies*, bei denen die Daten zunächst in verschlüsselter Form vorliegen

24 Manche Architekturen sehen einen Policy Retrieval Point (PRP) vor, welcher das Speichern und Abfragen von Policies ermöglicht [Bie17, S.109]

25 Für die Auflösung der Konsistenzprobleme ist in [Bie17] der übergeordnete PMP zuständig

Collection Point (ProCP) ist nur dann notwendig, wenn das Provenance Tracking verteilt mit mehreren, bestimmten Domänen zugewiesenen, ProSPs stattfindet.[26] Der Autor Bier schlägt in [Bie17] zusätzlich einen Provenance Dissemination Point (ProDP) vor, der Abfragen zur Provenance erlaubt und die Ergebnisse aufbereitet. Außerdem sind in der von ihm vorgestellten Architektur der PIP und ProSP miteinander gekoppelt, um zusätzliche Synergieeffekte, wie das Einbeziehen der Provenance in die Nutzungskontrollentscheidung, zu realisieren. Ebenso wie die Nutzungskontrolle setzt auch Provenance Tracking eine vertrauensvolle Umgebung voraus [IDS-RAM].

26 Hier zeigt sich wieder die hierarchische Architektur

3. Cloud Computing

Rechenleistung ist in den letzten Jahrzehnten omnipräsent geworden. Musste man sich in den Anfängen des Computerzeitalters noch über ein Terminal mit einem Großrechner verbinden, so haben heutzutage selbst Armbanduhren, wie die Apple Watch, ein Vielfaches deren Leistung. Da Rechenleistung jedoch erst durch Software nutzbar wird, zählen Entwicklung sowie Betrieb zu einer der größten Kostentreibern. Die Cloud verspricht, beides zu vereinfachen und die Kosten effektiv zu senken.

Dieses Kapitel dient der Aufbereitung des Themengebiets *Cloud Computing* und bildet damit die Grundlage für Kapitel 4, das sich mit der Zusammenführung von *Cloud Computing*-Technologien sowie Nutzungskontrolle und Provenance Tracking beschäftigt. Aus zeitlichen Gründen konnten nicht alle CSPs gleichermaßen betrachtet werden, weshalb die Google Cloud oftmals als Stellvertreter fungiert.

3.1. Definition

Die Definition von *Cloud Computing* ist nicht allgemeingültig festgelegt, weshalb in der Literatur viele unterschiedliche Variationen vorzufinden sind. Dies ist nicht zuletzt dem Umstand geschuldet, dass *Cloud Computing* eine hohe Dynamik aufweist und sich stetig weiterentwickelt. Die jedoch am häufigsten zitierte Definition ist die des National Institute of Standards and Technology (NIST). Darin heißt es:

> „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models“ [NIST-SP800-145, S.2]

Dagegen fällt die Definition des Bundesamt für Sicherheit in der Informationstechnik (BSI) pragmatischer aus:

> „Cloud Computing bezeichnet das dynamisch an den Bedarf angepasste Anbieten, Nutzen und Abrechnen von IT-Dienstleistungen über ein Netz. Angebot und Nutzung dieser Dienstleistungen erfolgen dabei ausschließlich über definierte technische Schnittstellen und Protokolle. Die Spannbreite der im Rahmen von Cloud Computing angebotenen Dienstleistungen umfasst das komplette Spektrum der Informationstechnik und beinhaltet unter anderem Infrastruktur (z. B. Rechenleistung, Speicherplatz), Plattformen und Software." [BSI-CC, S.15-16]

Jedoch steht die Definition des BSI nicht für sich alleine, sondern beinhaltet Konzepte, die in der Definition der NIST aufgegriffen werden. Daher wird im Folgenden der Begriff *Cloud Computing* anhand der NIST-Definition thematisiert. Die fünf essenziellen Eigenschaften von *Cloud Computing* sind:

On-demand Self Service Ressourcen können Nutzern bedarfsgerichtet und ohne menschliche Interaktion mit dem CSP bereitgestellt werden.

Broad Network Access Die Dienste eines CSP sind über das Netzwerk mit standardisierten Schnittstellen verfügbar. Nutzer sind in der Folge nicht an konkrete Clients gebunden.

Resource Pooling Ressourcen werden in Form eines Pools vom CSP zur Verfügung gestellt. Der genaue Standort der Ressourcen ist dem Nutzer unbekannt, kann aber vertraglich eingegrenzt werden (zum Beispiel auf bestimmte Rechenzentren oder Regionen). Da mehrere Nutzer Zugriff auf gemeinsame Ressourcen haben, müssen diese mandantenfähig sein.

Rapid Elasticity Die dem Nutzer zur Verfügung gestellten Ressourcen sind elastisch. Damit ist gemeint, dass die Bereitstellung und die Freigabe von Ressourcen schnell und gegebenenfalls automatisch stattfindet. Von außen betrachtet scheinen Anwendungen bis ins Unendliche zu skalieren.

Measured Service Die Nutzung von Ressourcen kann überwacht und gemessen werden. Da viele CSP ein *Pay per Use*-Modell anbieten, ergibt sich diese Eigenschaft von selbst.

Das BSI merkt hierzu an: Die Charakteristiken nicht zu dogmatisch zu sehen, da je nach Anwendungsfall auch unterschiedliche Priorisierungen vorliegen können [NIST-SP800-145][BSI-CC, S.15].

3.2. Servicemodelle

Servicemodelle geben die Art des *Services* an, welche CSP anbieten. Hierzu gehören:

Infrastracture as a Service (IaaS) Hier wird den Nutzern IT-Infrastruktur, zu denen Server (sowohl physisch als auch virtuell), Speicher und Netzwerk zählen, bereitgestellt. Die Nutzer können selbst entscheiden, welches Betriebssystem und welche Anwendungen darauf laufen.

Platform as a Service (PaaS) PaaS-Provider stellen Infrastruktur für den Betrieb von Anwendungen zur Verfügung. Der Kunde hat hierbei keinen direkten Zugriff auf das Betriebssystem oder die Hardware. Ihm wird also die Einrichtung sowie Verwaltung der zugrundeliegenden Infrastruktur abgenommen.

Software as a Service (SaaS) SaaS-Provider stellen Nutzern Anwendungen bereit. Jedwede Verwaltungsaufgaben (Infrastruktur, Wartung, Aktualisierung der Anwendung etc.) werden vom Provider übernommen.
[NIST-SP800-145; BSI-CC]

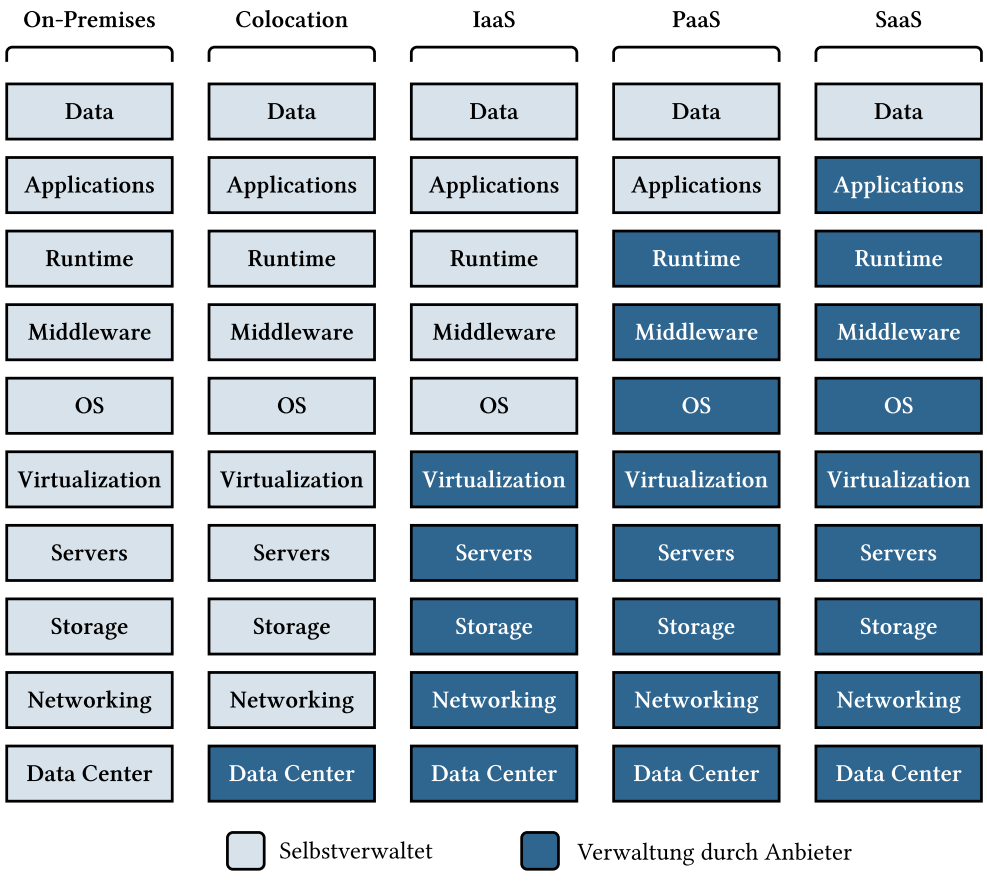

Abbildung 3.1.: Verwaltung der IT-Infrastruktur im Vergleich zu *On-Premises*, *Colocation* und den Servicemodellen [Hat]

Die Abbildung 3.1 verdeutlicht die Aufteilung der Verantwortlichkeiten je nach gewähltem Servicemodell. Die *Colocation* ist hierbei eine Form des *Outsourcing*, wobei sicherlich noch weitere Formen existieren. Unter dem Gesichtspunkt des *Outsourcing* sind die Servicemodelle nichts neues, lediglich die Umsetzung hat sich durch das *Cloud Computing* verändert. Der Begriff *Virtualization* schließt hierbei die *Network Virtualization* und *Virtual Machines* mit ein. *Middleware* dient der Kommunikation zwischen Anwendungen und der Punkt *Data* meint explizit nicht die technische Handhabung und Verwaltung von Daten (dies fällt in die Kategorie *Storage*), sondern bezieht sich auf die logische Verwaltung der Daten durch den Nutzer.

Zwischen PaaS und SaaS hat sich ein weiteres Servicemodell etabliert, welches nicht Teil der ursprünglichen Definition der NIST ist. *Serverless Computing* beschreibt eine Technik, bei der Anwendungen (ohne eigenen Zustand) nur nach bestimmten Ereignissen ausgeführt werden. Diese Anwendungen werden zumeist als Funktionen bezeichnet, weshalb sich der Begriff Functions as a Service (FaaS) etabliert hat.[1] Dies hat sowohl Kostenvorteile, da nur die tatsächlich aufgewendete Zeit für die Ausführung der Funktion bezahlt werden muss,[2] als auch Vorteile in der Skalierbarkeit, da zustandslose Anwendungen beliebig oft gestartet werden können. Auf diese Weise kann eine hohe Elastizität geboten werden. Elastizität meint hierbei eine automatische Anpassung der bereitgestellten Ressourcen nach dem tatsächlichen Bedarf [HKR13; Al+18]. Anwendungen, die *Serverless* betrieben werden, dürfen derzeit jedoch nur kurzlebig und nicht rechenintensiv sein.[3] Das heißt jedoch nicht, dass diese keinen Zustand ändern können, so gibt es bereits Datenbanken, wie FaunaDB,[4] die *serverless* arbeiten. Ein weiterer zu beachtender Punkt ist der sogenannte *Cold Start*, der genau dann eintritt, wenn für die Ausführung der Funktion eine neue Instanz erzeugt werden muss.[5] Die neuen Instanzen werden für darauffolgende Aufrufe „warm“ gehalten. Sollte kein weiterer Aufruf erfolgen, so werden die Ressourcen nach einer gewissen Zeit wieder freigegeben [Rob18].

3.3. Deployment-Modelle

Deployment-Modelle geben an, wo die Bereitstellung der Cloud-Infrastruktur erfolgt. Die vier möglichen Modelle sind:

1 Eine der bekanntesten Implementierungen ist AWS Lambda `https://aws.amazon.com/de/lambda` (besucht am 26. 10. 2020)

2 Messungen finden im Millisekundenbereich statt

3 AWS Lambda ist zu diesem Zeitpunkt auf 3GB RAM und maximal 15 Minuten Laufzeit limitiert `https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html` (besucht am 29. 10. 2020)

4 `https://fauna.com` (besucht am 05. 11. 2020)

5 Interpretierte Programmiersprachen, wie Python und NodeJS oder bereits compilierte wie Go und Rust, können schneller gestartet werden

Private Cloud *Cloud Computing*-Ressourcen werden exklusiv von einer einzigen Institution genutzt. Die *Private Cloud* kann sowohl von der Instutition selbst, als auch von einem dritten gehostet und/oder verwaltet werden. Gegenüber *On-Premises*-Lösungen bietet die *Private Cloud* mehr Flexibilität im Umgang mit den vorhandenen Ressourcen und eine bessere Skalierbarkeit.

Public Cloud Die Cloud-Infrastruktur kann von der Allgemeinheit genutzt werden, wobei diese mitsamt der Software Eigentum des CSP ist und auch von diesem verwaltet wird. *Public Clouds* haben zusätzlich den Vorteil geringer Kosten und einer enormen Skalierbarkeit sowie Zuverlässigkeit.

Community Cloud Die Cloud-Infrastruktur wird von mehreren Institutionen geteilt. Die Verwaltung der Cloud kann von den Institutionen selbst oder einem Dritten erfolgen.[6]

Hybrid Cloud Hierbei werden mehrere eigenständige Cloud-Infrastrukturen über standardisierte Schnittstellen miteinander verbunden. Dies ermöglicht Institutionen eine Skalierung über die eigene Infrastruktur hinaus, ohne zusätzliche Investitionen in diese tätigen zu müssen. Sensible Daten können in der eigenen *Private Cloud* verbleiben, während bestimmte Aufgaben in anderen Clouds ausgeführt werden. Damit bietet die *Hybrid Cloud* einen Kompromiss zwischen *Private* und *Public Cloud*, in der Wert aus einer bereits vorhandenen Infrastruktur geschöpft werden kann. Die Kontrolle über die eigenen Daten kann bestehen bleiben und bestimmte Arbeiten können gezielt ausgelagert werden, wodurch eine insgesamt hohe Kosteneffizienz möglich ist.
[BSI-CC; MS-AzureA; MS-AzureB]

3.4. Aspekte der Sicherheit und des Rechtlichen

Sicherheit in der Cloud grenzt sich nur bedingt zur Sicherheit in „traditionellen“ IT-Systemen ab. So sollte in beiden Fällen die Auswahl geeigneter Sicherheitsmaßnahmen anhand der Anforderungen sowie des konkreten Verwendungszwecks erfolgen. Bezogen auf die Cloud beschreibt das BSI den Sachverhalt wie folgt:

> „So wie es nicht das eine sichere Auto für alle Situationen gibt, so gibt es auch nicht die eine sichere Cloud für alle Fälle. [...]
> Das Ziel lautet daher: Sicheres *Cloud Computing* und nicht die sichere Cloud. Letztere gibt es nicht.“ [BSI-SNC, S.6]

Daher ist der Weg in die Cloud mit einem hohen Planungsaufwand, der sogenannten „Cloud-Strategie“, verbunden. In dieser werden die grundlegenden Anforderungen sowie Rah-

6 Als Beispiel sei das Europäische Großprojekt GAIA-X genannt [GAIA-X]

menbedingungen ermittelt, um daraus konkrete Handlungen abzuleiten. Hierzu gehören unter anderem eine Analyse der Sicherheitsanforderungen und Risiken sowie ein Plan zum Ausstieg aus der Cloud [BSI-SNC].

3.4.1. Vertrauen

Um das Vertrauen in IT-Systeme zu stärken, existieren eine Reihe von Zertifizierungsprozessen. Der in Deutschland bekannteste ist der IT-Grundschutz des BSI, welcher kompatibel zum internationalen Standard ISO 27001 ist. Dessen Ziel ist die Identifikation sowie Umsetzung von geeigneten Sicherheitsmaßnahmen zur Erreichung eines angemessenen Schutzniveaus. Durch Zertifizierungen können sich Organisationen die Erfüllung von bestimmten Anforderungen von externen (unabhängigen) Auditoren bescheinigen lassen, wodurch das Vertrauen in die Organisation gestärkt wird. Speziell für die Anforderungen an die Cloud hat das BSI den Cloud Computing Compliance Criteria Catalogue (BSI C5) etabliert, welcher ein Sicherheitsniveau mit erforderlichen Nachweisen beschreibt, das von CSPs nicht unterschritten werden soll. Zertifizierungen helfen auch dabei, mehr Transparenz zwischen Nutzer und CSP zu schaffen, da auf diese Weise die Rahmenbedingungen des CSP klarer werden. Dadurch wird der Nutzer befähigt, seine rechtlichen Anforderungen mit den Rahmenbedingungen des CSP abzustimmen und so zu erkennen, welche Anforderungen, und damit auch Aufgaben, in seinen Verantwortlichkeitsbereich fallen und ob die Summe der Anforderungen durch beide erfüllt werden.[7] Dieses Modell der geteilten Verantwortlichkeiten wird als *Shared Responsibility Model* bezeichnet. Schwierigkeiten ergeben sich vor allem dadurch, dass je nach gewähltem Servicemodell, Deployment-Modell und damit auch den CSPs, Unterschiede in der Verteilung der Verantwortlichkeiten bestehen [BSI-CC; BSI-SNC; CSA-SRM].

3.4.2. Datenschutz

Besonderes Augenmerk lässt sich auf den Punkt Datenschutz legen, welcher historisch bedingt in Deutschland einen hohen Stellenwert einnimmt. Die in der EU verbindliche Verordnung ist hierfür die Datenschutz-Grundverordnung (DSGVO), engl. General Data Protection Regulation (GPDR).[8] Die DSGVO betrifft jedoch ausschließlich personenbezogene Daten. Daten welche keinen Bezug zu einer natürlichen Person haben, fallen nicht unter das Datenschutzrecht, glei-

7 Hierzu stellen die Cloud-Anbieter i. d. R. Übersichten auf deren Website bereit. Für AWS: `https://aws.amazon.com/de/compliance/programs` (besucht am 02. 11. 2020),
Google Cloud: `https://cloud.google.com/security/compliance` (besucht am 02. 11. 2020) und Azure: `https://azure.microsoft.com/de-de/overview/trusted-cloud/compliance` (besucht am 02. 11. 2020)

8 Die DSGVO wurde in Deutschland durch die Version 2.0b des Standard-Datenschutzmodell (SDM) in technische sowie organisatorische Maßnahmen überführt [SDM]

ches gilt für anonymisierte Daten. Die DSGVO umfasst Dokumentations-, Organisations- und Transparenzpflichten. Ziel dieser Pflichten ist eine Auseinandersetzung mit der Verarbeitung von personenbezogenen Daten. Eine Zuwiderhandlung kann mit hohen Bußgeldern bestraft werden. Die in der DSGVO beschriebenen Pflichten betreffen ausnahmslos jedes Unternehmen und sind keine cloudspezifischen Anforderungen. Im Zusammenhang mit der Nutzung von *Cloud Services* ist Art. 28 (Auftragsverarbeitung) der DSGVO maßgeblich. Demnach ist der Auftragsverarbeiter weisungsgebunden gegenüber dem Auftraggeber, mit dem er im Vorfeld eine Vereinbarung über die Auftragsverarbeitung geschlossen haben muss. Ferner muss der Auftraggeber sicherstellen, dass der Auftragsverarbeiter (in diesem Fall der CSP) die datenschutzrechtlichen Anforderungen erfüllt. Dies muss er jedoch nicht selbst tun, sondern kann in Form einer Zertifizierung erfolgen. Es müssen sowohl der Auftragsverarbeiter als auch der Auftragsgeber die Anforderungen an die DSGVO erfüllen, ist dies nicht der Fall, so sind theoretisch beide haftbar. In der Praxis könnte es jedoch schwierig sein, einen Haftungsanspruch gegenüber dem Auftragsverarbeiter geltend zu machen. Darüber hinaus müssen die Rechte der betroffenen Personen gewahrt bleiben. Eines dieser Rechte ist, dass die Verarbeitung der personenbezogenen Daten auf eine nachvollziehbare Weise erfolgen muss. Hierfür muss der Betroffene der Verarbeitung vorher zugestimmt haben oder eine andere Rechtsgrundlage geltend gemacht werden. Ferner muss eine angemessene Sicherheit der Daten gewährleistet sein, die sich an dem „Stand der Technik“ orientiert [Hub18; ECO-DSGVO].

Ein weiterer Punkt ist Art. 25, der *Privacy by Design* sowie *Privacy by Default* vorsieht. Eine Übermittlung von personenbezogenen Daten in Drittländer oder an internationale Organisationen ist nur dann zulässig, wenn das Schutzniveau für die betroffene Person dadurch nicht untergraben wird oder es ein entsprechendes Abkommen mit der EU gibt.

Das Problem ist, dass selbst mit den USA, in denen die größten CSPs ihren Sitz haben, zur Zeit kein solches Abkommen existiert. Sowohl das *Safe Harbour*-Abkommen, als auch das *Privacy Shield* wurden vom Europäischen Gerichtshof als nichtig erklärt. Das heißt, dass jedes Unternehmen mit Sitz in den USA ein nicht angemessenes Datenschutzniveau aufweist und somit eine Übermittlung von personenbezogenen Daten unrechtsmäßig ist. Ein Grund hierfür ist, dass Unternehmen in den USA, selbst wenn sie international agieren, immer noch dem US-Recht unterstehen [Bec20; Kre20].[9]

3.4.3. Sicherheit

IT-Sicherheit baut traditionsgemäß auf physischen Barrieren auf. Durch die Cloud verschwimmen die Grenzen solcher Barrieren, da – zumindest bei Nutzung eines CSPs – Ressourcen unter

9 Siehe hierzu den Clarifying Lawful Overseas Use of Data Act (CLOUD Act)

mehreren Nutzern geteilt werden und kein physischer Zugriff auf die Infrastruktur möglich ist. Das heißt jedoch explizit nicht, das keine physischen Barrieren vorhanden sind. Ganz im Gegenteil, Stephanie Wong zeigt hierzu in [Won20] das Sechs-Schichtenmodell für die physische Sicherheit in Google-Rechenzentren.[10] Die darin beschriebenen Maßnahmen und resultierende Sicherheit liegt oft weit über dem, was Betreiber von *On-Premises*-Lösungen oder *Private Clouds* zumutbar beziehungsweise finanziell aufwendbar ist. Hier liegt ein ganz klarer Vorteil in der Benutzung von CSPs: Während im eigenen Unternehmen das Budget für IT und IT-Sicherheit restriktiv gehandhabt werden kann, können sich CSPs hochgradig spezialisieren [Dot19, S. 9]. So bieten CSPs, wie Google, Disaster Recovery Plans (DRPs) an, um auf Naturkatastrophen sowie Hardwarefehler, menschlichen Fehlern oder auch Computerkriminalität entsprechend zu reagieren [Seh20; Google19W].

3.4.3.1. Data Asset Management

Dotson [Dot19] teilt das Management der Cloud in zwei Bereiche auf, dem *Data Asset Management* und dem *Cloud Asset Management*. *Data Asset Management* behandelt den Umgang mit Daten, wie Kundendaten in der Cloud. Risiken die hierbei auftreten sind: (i) Die Veröffentlichung sowie der Verlust von Daten, (ii) *Vendor Lock-in*, sollten die Daten nicht vollständig exportierbar sein und (iii) residuale Daten, womit Daten bezeichnet werden, die noch im Speicher liegen und nicht überschrieben wurden [Seh20]. Um die Risiken im Umgang mit Daten abzuschätzen, ist es notwendig, eine Klassifizierung vorzunehmen. Diese Klassifizierung kann anhand der Schwere des Verlusts oder der Schwere einer Veröffentlichung der Daten erfolgen. CSPs bieten mit Cloud Data Loss Prevention (DLP) Tools an, die eine Klassifizierung und gegebenenfalls Maskierung von Daten ermöglichen.[11] Daten können drei mögliche Zustände annehmen: (a) *in motion*, (b) *in use* und (c) *at rest*. Für Daten *in motion* und *at rest* bieten CSPs in der Regel eine automatische Verschlüsselung an [Google19W].

Die Verschlüsselung der zu übertragenden Daten geschieht über Transport Layer Security (TLS) oder mutual TLS (mTLS), wobei letzteres eine gegenseitige Authentifizierung der Kommunikationspartner ermöglicht.[12] Diese bidirektionale Authentifizierung dient vor allem dem Datenaustausch zwischen Anwendungen, bei der sichergestellt werden muss, dass sowohl Anfragender, als auch der Angefragte ein gültiges Zertifikat besitzen. Im Kubernetes-Cluster kann hierfür Istio verwendet werden, das die Zertifikatsverwaltung für mTLS automatisiert.[13]

Auch die Verschlüsselung von Daten, welche in der Cloud gespeichert werden, ist bei

10 Diese Form der Sicherheit ist in der Regel auch Bestandteil von Zertifizierungen
11 `https://cloud.google.com/dlp` (besucht am 05. 11. 2020)
12 mTLS ist kein eigenständiges Protokoll, sondern optional über TLS aktivierbar
13 `https://istio.io/latest` (besucht am 06. 11. 2020)

vielen CSPs standardmäßig aktiviert. Darüber hinaus ermöglichen manche CSPs den Nutzern eine feingranulare Verwaltung der Schlüssel, die zur Verschlüsselung verwendet wurden. Google bietet hierfür den Cloud Key Management Service (KMS) an, um eigene Schlüssel zu erzeugen, zu rotieren und zu zerstören.[14] Diese Form der Schlüsselverwaltung wird auch als Customer Managed Encryption Keys (CMEKs) bezeichnet. KMSs nutzen hierfür Hardware Security Modules (HSMs).[15] Dabei handelt es sich um hochspezialisierte Hardware, die eine sichere Verwaltung der Schlüssel ermöglicht sowie performant kryptographische Funktionen ausführen kann. Die Benutzung eines solchen HSM ist keinesfalls einfach, weshalb die Nutzung eines KMS einfacher und sicherer ist, auch wenn man nun sowohl KMS als auch HSM vertrauen muss. Ferner sind HSMs nicht günstig [Han+19]. Unternehmen deren Budget für den Kauf eines HSMs nicht ausreicht und damit auf deren Funktionalität verzichten müssten, können diese stattdessen über die Cloud nutzen. Auch gibt es die Möglichkeit, das Kunden ihre eigenen Schlüssel nutzen, auch Customer Supplied Encryption Keys (CSEKs) gennant. Zum Zeitpunkt der vorliegenden Arbeit befindet sich Cloud External Key Manager (EKM) von Google in der Beta-Phase, wodurch Kunden die Nutzung eines externen KMS möglich ist. Dadurch behält der Nutzer die Kontrolle über Erzeugung, Ort, Verteilung und Verwendung der Schlüssel [Lee19].[16] Verschlüsselung ermöglicht das kryptographische Löschen. Anstelle die Daten selbst zu löschen, wird der Schlüssel, der zur Verschlüsselung verwendet wurde, gelöscht. Dabei stützt man sich auf die Annahme, dass die verschlüsselten Daten nur mit dem ursprünglichen Schlüssel wiederherstellbar sind. Hierbei werden oft zwei Schlüssel verwendet, ein sogennanter *Key Encryption Key* und ein *Data Encryption Key*. Der *Data Encryption Key* wird zum Verschlüsseln der Daten verwendet, während der *Key Encryption Key* zum Verschlüsseln des *Data Encryption Keys* verwendet wird. Dadurch kann der *Data Encryption Key* direkt neben den verschlüsselten Daten gespeichert werden.[17] Die bislang angesprochene Verschlüsselung der Daten *at rest* geschah auf Seite des Servers. Es existiert aber auch die Möglichkeit, die Daten bereits auf Seite des Clients zu verschlüsseln. Dadurch wird jedoch die Verarbeitung der Daten auf Seite des Servers stark eingeschränkt, da dieser den verwendeten Schlüssel nicht kennt. Hierbei kommen Verfahren wie homomorphe Verschlüsselung zum Einsatz, welche jedoch noch weitere Forschung benötigen und an dieser Stelle nicht weiter vertieft werden sollen [Google19W; Dot19, S. 14-20].

14 Im Zuge der Nutzung eines KMS sollte man sich mit *Break Glass*-Mechanismen auseinandersetzen, für den Fall das der KMS nicht erreichbar ist

15 HSMs meinen in diesem Kontext nicht USB-Tokens, Smartcards, Software HSMs o. Ä.

16 Weitere Informationen zum Thema *Key Management in Cloud Services* kann in [Ega+20] nachgelesen werden

17 Für weitere Information: `https://cloud.google.com/security-key-management` (besucht am 06. 11. 2020) und `https://cloud.google.com/security/encryption-at-rest/customer-supplied-encryption-keys` (besucht am 06. 11. 2020)

Neben der kryptographischen Löschung von Daten wird bei der konventionellen Löschung eine *Data deletion pipeline* angestoßen. Google organisiert diese *Pipeline* in vier Stufen: (i) *Deletion Request* (ii) *Soft Deletion*, (iii) *Logical Deletion* und (iv) *Expiration from Backup-Systems*. In der ersten Stufe werden zu löschende Daten markiert und in der zweiten werden diese dann formal entfernt, sind aber noch wiederherstellbar. Erst nach einem gewissen Zeitraum werden die Daten durch Überschreiben entfernt. Das betrifft die sogenannten aktiven Systeme, Google gibt hierfür eine maximale Dauer von zwei Monaten an. Für die Backup-Systeme, also Stufe vier, kann die Dauer bis zu sechs Monate betragen, wobei durch Überschreiben der Backups sowie kryptographisches Löschen die Zeit reduzierbar ist [Google19D].[18]

Der Schutz von Daten *in use* ist wesentlich schwieriger. Zum einen müssen die Anwendungen effektiv voneinander isoliert sein, sodass keine Anwendung auf den Speicher einer anderen zugreifen kann. Zum anderen dürfen die Anwendungen (attestierbar) nicht manipuliert worden sein. Der letzte Punkt stellt sicher, dass Daten wie vom Anwendungsentwickler vorgesehen verarbeitet werden. Ein Schutz, der beide Anforderungen erfüllt, realisiert *Confidential Computing*. Eine der prominentesten Technologien für *Confidential Computing* ist Intel SGX, welches sogenannte Enklaven für Anwendungen erstellt. Diese Enklaven werden effektiv geschützt, unter anderem durch Verschlüsselung des Speicherbereichs der Anwendung. Darüber hinaus ist es möglich, Anwendungen zu attestieren, um deren Integrität zu überprüfen. *Confidential Computing* bietet damit Eigenschaften, die durch HSMs realisiert wurden, jedoch wesentlich günstiger und mit einer höheren Geschwindigkeit, da nicht mehr PCIe oder das Netzwerk, sondern die CPU-Geschwindigkeit den Takt vorgibt. Bereits jetzt bieten CSPs, wie IBM mit *IBM Cloud Data Shield* für Kubernetes,[19] *Confidential GKE Nodes* für Kubernetes[20] und *Confidential Virtual Machines (VMs)*[21] von Google oder Azure, sowohl für VMs[22] als auch für Kubernetes[23] die Möglichkeit, *Confidential Computing* in der Cloud zu nutzen.[24] Ferner ist es mit Scone möglich, auch im selbstverwalteten Kubernetes-Cluster *Confidential Computing* einzusetzen.[25] Auf die Vorzüge, hierbei auf Kubernetes zu setzen, wird im nächsten Kapitel eingegangen [Seh20; Bra20].

18 Weitere Information zur *Data deletion pipeline* können im zitierten *Paper* [Google19D] nachgelesen werden
19 `https://www.ibm.com/cloud/data-shield` (besucht am 06. 11. 2020)
20 `https://cloud.google.com/kubernetes-engine/docs/how-to/confidential-gke-nodes` (besucht am 06. 11. 2020)
21 `https://cloud.google.com/compute/confidential-vm/docs` (besucht am 07. 11. 2020)
22 `https://docs.microsoft.com/de-de/azure/confidential-computing/confidential-computing-enclaves` (besucht am 06. 11. 2020)
23 `https://docs.microsoft.com/de-de/azure/confidential-computing/confidential-nodes-aks-overview` (besucht am 06. 11. 2020)
24 Für Microsoft Azure war wohl der *JEDI Contract* mit dem U.S. Defense Department Grund für das Bereitstellen von *Confidential Computing Services*
25 `https://sconedocs.github.io` (besucht am 06. 11. 2020)

Confidential Computing ist nicht mit *Trusted Computing* zu verwechseln. Der Unterschied liegt grob darin, dass beim *Trusted Computing* die Plattform, auf der der Code ausgeführt wird, sich in einem sicheren Zustand befinden soll und nicht nur die darauf laufende Anwendung. Während zum Zeitpunkt der Arbeit *Confidential Computing* nicht durch offene Standards realisiert werden kann, muss den Prozessorherstellern wie Intel vertraut werden. Hierbei ist vor allem wichtig, das Sicherheitslücken gemeldet und entsprechend korrigiert werden.[26] Beim *Trusted Computing* handelt es sich mit den TPMs, welche als Vertrauensanker fungieren, um eine offene Spezifikation der Trusted Computing Group (TCG).[27]

3.4.3.2. Cloud Asset Management

Der nächste zu betrachtende Punkt ist *Cloud Asset Management*, womit alle Cloud-Ressourcen gemeint sind, die Daten speichern oder verarbeiten. Grundlage eines sicheren Umgangs mit Cloud-Ressourcen bildet ein wirksames Identity and Access Management (IAM), in der Nutzer nur die absolut notwendigsten Rechte haben (*Principle of least privilege*). Cloud-Ressourcen werden in der Regel über eine *Cloud Console* erzeugt, geändert oder freigegeben. Die *Cloud Console* kann über eine Webseite des CSP aufgerufen werden, oder aber über eine enstprechende Terminal-Anwendung. Der Zugriff auf diese Konsole ist so schützenswert wie der physische Zugriff im Rechenzentrum. Aus diesem Grund sollte eine Multi-Faktor-Authentisierung genutzt werden [Dot19, S. 1]. Auch der Zugriff auf die Cloud-Ressourcen muss abgesichert werden. Auf Netzwerkebene kann hierzu eine Virtual Private Cloud (VPC) genutzt werden, wodurch sich Ressourcen zu einem virtuellen Netzwerk verbinden lassen. Darüber hinaus können durch Identity-Aware Proxies (IAPs) Zugriffe auf Ressourcen und Anwendungen anhand von Identität und Kontext weiter limitiert werden – neben klassichen Verfahren wie Virtual Private Network (VPN). Zugriffe und Anwendungslogs können in der Cloud nahezu in Echtzeit überwacht werden. Dies ermöglicht ein Etablieren von *Intrusion Detection Systems*, welche bei bestimmten Ereignissen Benachrichtigungen auslösen, sowie *Intrusion Prevention Systems*, die Zugriffe auf Ressourcen blockieren können [Dot19, S. 133-134].

Attacken auf VMs lassen sich in zwei Kategorien unterteilen. Die erste sind Attacken auf den *Hypervisor* der VM. Diese Angriffsmethode ist jedoch wenig erfolgsversprechend, da speziell „gehärtete“ *Hypervisor* zum Einsatz kommen. Die zweite Methode greift die VM über einen Seitenkanal (engl. *Side-Channel*) an. Dies kann zum Beispiel eine andere VM sein,

26 In der Vergangenheit kam es hierbei immer wieder zu Problemen (s. Spectre und Meltdown). Vor allem Intels *Management Engine* weist regelmäßig Sicherheitslücken auf. Erst vor Kurzem wurde eine Sicherheitslücke in Intel SGX gefunden [Sch+20]

27 Microsoft kündigte zusammen mit Intel, AMD und ARM den Microsoft-Controller „Pluton“ an, der in deren Prozessoren verankert und „Chip-to-Cloud Security“ bieten soll [MS-Win20A; MS-Win20B]

welche auf demselben physischen Server ausgeführt wird.[28] Um Attacken über Seitenkanäle vorzubeugen, kann entweder eine *dedicated* VM genutzt werden, bei der sichergestellt ist, dass nur eine VM darauf läuft oder aber *Bare-Metal Solutions*, wobei ein echter physischer Server ohne Virtualisierung genutzt wird. Auch die Verschlüsselung des Arbeitsspeichers stellt einen wirksamen Schutz gegen diese Form von Angriff dar. Ein durch die Cloud hinzukommender Angriffsvektor betrifft Funktionalitäten, die CSPs als PaaS- und IaaS-Provider bereistellen. Darin könnten sich theoretisch mögliche Schwachstellen befinden. Dem Autor der vorliegenden Arbeit ist jedoch kein konkreter Fall einer solchen Schwachstelle bekannt. Die meisten Fehler in der Cloud-Nutzung sind auf fehlerhafte Konfigurationen zurückzuführen [Gartner20E]. Allerdings ist die stetige Zunahme an Denial of Service (DoS)-Attacken besorgniserregend, wodurch Systeme für Minuten, Stunden oder sogar Tage außer Gefecht gesetzt werden [Seh20].

Ebenso schützenswert wie Komponenten der Infrastruktur sind die Anwendungen, die auf dieser ausgeführt werden. Hierzu bietet das Open Web Application Security Project (OWASP) eine jährlich aktualisierte Liste der zehn häufigsten Schwachstellen in Anwendungen an. Nicht weniger kritisch sind Schwachstellen in Abhängigkeiten der Anwendungen, wie der *Heartbleed Bug* in OpenSSL zeigte. Aus diesem Grund sollten verwendete Abhängigkeiten immer aktuell gehalten und auf Common Vulnerabilities and Exposures (CVEs) überprüft werden. Die meisten Anwendungen werden nicht mehr direkt auf dem Betriebssystem, sondern in Containern ausgeführt. Die Verwendung von Containern hat viele operative Vorteile, aber auch Nachteile, da die Isolation von Containern gegenüber VMs wesentlich geringer ist und dadurch eine größere Angriffsfläche entsteht. Die Betrachtung der Container-Sicherheit würde jedoch den Rahmen dieser Arbeit sprengen, weshalb auf weiterführende Literatur, wie [Ric20] und [NIST-SP800-190], verwiesen werden muss. Generell befasst sich Container-Sicherheit mit dem Schutz des Host-Betriebssystems und dessen Ressourcen sowie dem Schutz der Anwendung von außen.

Werden Anwendungen mit Kubernetes bereitgestellt, so muss Kubernetes selbst abgesichert werden. Eine Checkliste für zutreffende Maßnahmen stellt der *CIS Kubernetes Benchmark* bereit.[29] Google bietet darüber hinaus weitere Sicherheitsmechanismen an. *GKE Sandbox* verwendet gVisor, um den Kernel des Host-Betriebssystems zu schützen und eine zusätzliche Isolationsschicht zwischen Container und Betriebssystem zu schaffen.[30] *Shielded GKE Nodes* sind Nodes im Kubernetes Cluster, bei denen mittels TPMs kryptographisch sichergestellt ist, dass diese in einem Google Rechenzentrum stehen und sicher in ihr Betriebssystem gebootet haben (*Plattform integrity*). Des Weiteren stellen sie dem Nutzer einen virtuellen TPM bereit.

28 Meltdown und Spectre sind solche Beispiele

29 `https://www.cisecurity.org/benchmark/kubernetes` (besucht am 06. 11. 2020)

30 `https://gvisor.dev` (besucht am 06. 11. 2020)

Mithilfe von Binärautorisierung wird die Signatur eines Containers vor dessen Bereitstellung im Kubernetes-Cluster überprüft [Google20-GKE].[31] Weitere Maßnahmen umfassen die Limitierung von Ressourcen eines Nodes sowie Kontrolle über die Privilegien, mit denen die Container auf dem Host-Betriebssystem ausgeführt werden [Seh20; Dot19; Google19-CS].

31 `https://cloud.google.com/binary-authorization` (besucht am 06. 11. 2020)

4. Nutzungskontrolle und Provenance Tracking in der Cloud

Die in Kapitel 2 und 3 gesammelten Erkenntnisse sollen im Folgenden zur Ausbringung einer prototypischen Nutzungskontroll- sowie Provenance sammelnden Infrastruktur genutzt werden.

Konkret lauten die Ziele: (i) Vereinfachung der Kommunikation zwischen den Komponenten der in 2.3.2.1 und 2.3.2.2 beschriebenen Referenzarchitektur, (ii) Verbesserung der operativen Aspekte, insbesondere des Konfigurationsmanagements, sowie (iii) die Erforschung von Möglichkeiten zur automatisierten Bereitstellung einer solchen Infrastruktur. Die Basis zur Erreichung dieser Ziele wird Kubernetes sein, das in Abschnitt 4.3 besprochen wird. Die darauffolgende Modellierung und Implementierung ist nicht statisch, sondern wird anhand unterschiedlicher Reifegrade bezüglich eines *Cloud Native*-Ansatzes betrachtet.

4.1. Cloud Native

Cloud Native beschreibt ein Paradigma, in dem sowohl technisch, organisatorisch als auch kulturell auf den optimalen Betrieb in einer *Cloud Computing*-Umgebung hingearbeitet wird. Die Cloud Native Computing Foundation (CNCF)[1] – ein von der Linux Foundation gestartetes Projekt, welches Entwicklungen hinsichtlich *Cloud Native*-Technologien überwacht und koordiniert[2] – definiert *Cloud Native* wie folgt:

> „Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. [...] These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil. [...]“ [CNCF18]

1 `https://www.cncf.io` (besucht am 19. 11. 2020)
2 Eine Übersicht aller CNCF-Projekte ist hier zu finden `https://landscape.cncf.io` (besucht am 19. 11. 2020)

Die Motivation zur Anwendung eines solchen Paradigmas liegt in einer schnelleren Entwicklung und Bereitstellung von Software, einer erhöhten Skalierbarkeit und Leistungsfähigkeit sowie einer verbesserten Verwaltung komplexer verteilter Systeme [RDG19, S. 1-11]. Software, die nach dem *Cloud Native*-Paradigma konstruiert wurde, ist von Natur aus verteilt und als Menge von lose gekoppelten *Services* realisiert. Solche *Services* sind per Design schnell, robust und hochgradig skalierbar, nur *serverless* bereitgestellte Anwendungen können dieses Maß an Skalierbarkeit übertreffen [RDG19, S. 249-250, 286–289].

4.2. Systemkontext

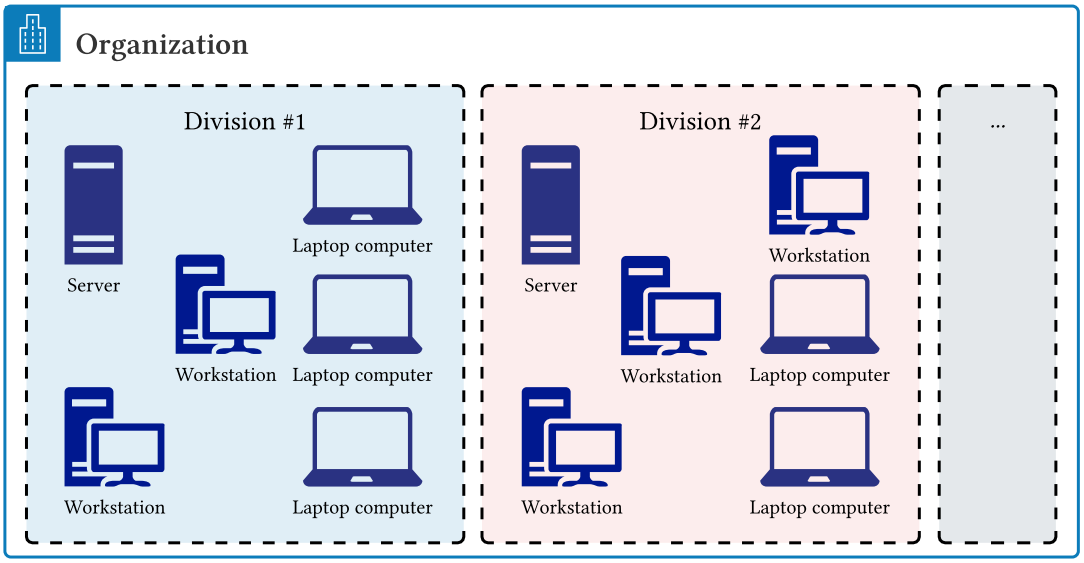

Abbildung 4.1.: Zu betrachtender Systemkontext

Im Rahmen dieser Arbeit beschränken wir uns zunächst auf die Betrachtung einer fiktiven Organisation mit jeweils unterschiedlichen Organisationseinheiten (siehe Abbildung 4.1). Die jeweiligen Organisationseinheiten sind mit Computern ausgestattet, die sich im organisationseigenen Netzwerk befinden. Des Weiteren existiert die Möglichkeit, für jeden Bereich einen (physischen oder virtuellen) Server bereitzustellen. Die Bereitstellung des Servers muss nicht lokal in dem jeweiligen Bereich erfolgen, sondern kann in einem zentralen Rechenzentrum stattfinden. Geräte außerhalb des Netzwerks können über VPN Zugang zum Organisationsnetzwerk erlangen. Das Organisationsnetzwerk kann Teil einer Cloud sein. Verteilte Nutzungskontrolle und Provenance Tracking meint in diesem Kontext also die Verteilung auf unterschiedliche Bereiche innerhalb einer logischen Organisation. Der Fall einer organisationsübergreifenden

Nutzungskontrolle und Provenance Trackings wird im späteren Verlauf der Arbeit diskutiert. Ferner wird angenommen, dass auf den jeweiligen Geräten (inlusive Server) Kubernetes installiert und entsprechend konfiguriert wurde. Die Geräte bilden also einen Kubernetes-Cluster.

4.3. Kubernetes

Kubernetes (abgekürzt K8s) ist eine Open Source-Plattform zur Orchestrierung von containerbasierten Anwendungen. Ein Container-Orchestrator ist ein Dienst, der die Aufgaben *Scheduling, Orchestration* und *Cluster Management* übernimmt. Als *Scheduling* wird das Management vorhandener Ressourcen und die Zuweisung von Arbeitslasten auf *Nodes* im Cluster bezeichnet. *Orchestration* ist die Koordinierung von Aktivitäten und Abläufen im Cluster und *Cluster Management* meint das Verwalten physischer oder virtueller *Nodes* [AD19, S. 10-11].

4.3.1. Hintergrund

Google begann bereits sehr früh damit, seine eigenen Dienste containerbasiert bereitzustellen. Zur Orchestrierung wurde ein System mit dem Namen Borg entwickelt. Dieses System war jedoch stark an Googles Interna und proprietäre Technologien gekoppelt, sodass nachträgliche Erweiterungen oder die Veröffentlichung des Quellcodes ausgeschlossen waren. Im Zuge dessen versuchte man die jahrelang gesammelten Erfahrungen der Containerorchestrierung mit Borg und die Ideen sowie Praktiken der Open Source Community in ein neues, quelloffenes Projekt einfließen zu lassen: Kubernetes [AD19, S. 1-14].[3,4]

Kubernetes war das erste Projekt das Teil der CNCF wurde. In den letzten Jahren hat Kubernetes den Status eines Industriestandards angenommen und bildet das Fundament für Plattformen wie OpenShift von RedHat. Jeder große CSP bietet mittlerweile Kubernetes als *Managed Service* an. In diesem Kontext wird Kubernetes oft als *Portability Layer* bezeichnet, da sich die Kubernetes-API – abgesehen von unterschiedlichen Versionen – zwischen den CSPs nicht ändert [SSJ19, S. 13-17].

4.3.2. Architektur

Kubernetes basiert auf einer ressourcenbasierten deklarativen API [IHD20, S. 175]. Vereinfacht ausgedrückt beschreibt man mithilfe von YAML-Dateien (Manifeste gennant) Kubernetes-Objekte, die den gewünschten Systemzustand beschreiben. Kubernetes versucht mithilfe des sogennanten *Reconciliation Loop*, den aktuellen Zustand an den gewünschten anzugleichen.

3 `https://github.com/kubernetes/kubernetes` (besucht am 20. 11. 2020)

4 Die Bezeichnung Kubernetes stammt aus dem griechischen und bedeutet soviel wie „Kapitän“

Die deklarative Beschreibung eines Systemzustandes erleichtert in enormen Maße das Konfigurationsmanagement. Zum einen können die Manifeste in die Versionskontrolle eingebucht werden, zum anderen kann der Systemzustand einfacher überblickt werden. Eine imperative Beschreibung würde den Systemzustand als Folge mehrerer Zustandsübergänge beschreiben, im Gegensatz zur deklarativen Beschreibung [BBH19, S. 1-11].

Ein Kubernetes-Cluster setzt sich aus einer Menge an Computern, auch *Nodes* genannt, zusammen. Ein Cluster besteht aus mindestens einem *Node*. Diese unterteilen sich in *Master Nodes* und *Worker Nodes*. *Worker Nodes* dienen als Host für *Pods*, welche aus einem oder mehreren Containern bestehen. Ein *Pod* ist die kleinste Einheit, die mit Kubernetes bereitgestellt werden kann. Dies hat zur Folge, dass alle Container eines *Pods* immer auch auf demselben *Node* bereitgestellt werden [IHD20, S. 1-11].

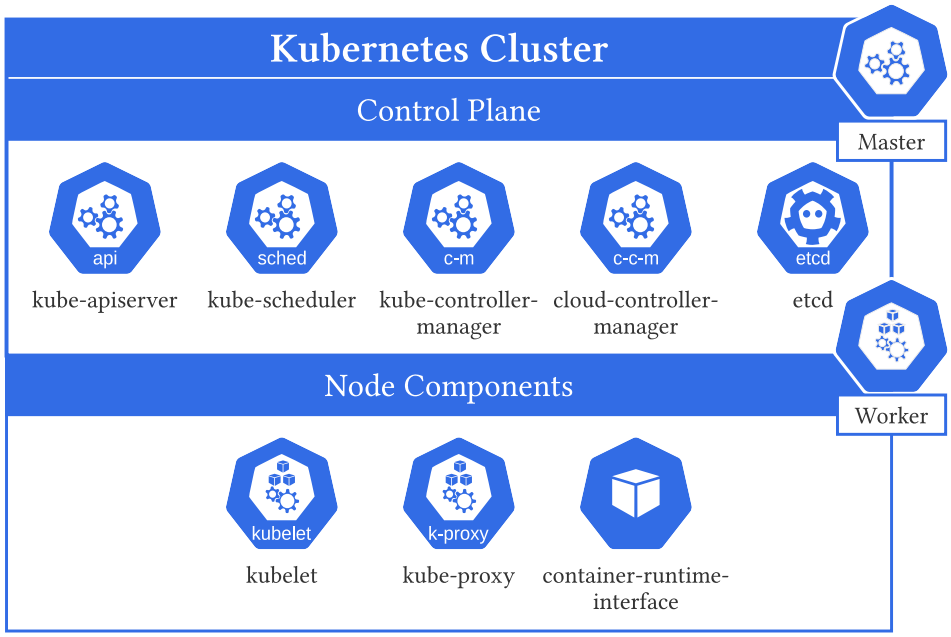

Abbildung 4.2.: Zentrale Komponenten eines Kubernetes-Clusters

Die *Master Nodes* eines Clusters bilden dessen *Control Plane* (siehe Abbildung 4.2). Die Aufgabe der *Control Plane* ist, Entscheidungen für den gesamten Cluster zu treffen und auf Events im Cluster zu reagieren. Der `kube-apiserver` ist die Schnittstelle zur Kubernetes-API und dessen Ressourcen. `etcd` ist ein verteilter *Key Value Store* mit starken Konsistenzbedingungen, in der alle für die Kubernetes-API relevanten Daten liegen. Der `kube-scheduler` ist für die Zuweisung von *Pods* auf *Nodes* zuständig. *Controller* sind Prozesse, die kontinuier-

lich den Zustand von Kubernetes-Ressourcen überwachen und Anpassungen zur Erreichung des Soll-Zustandes vornehmen. Der `kube-controller-manager` verwaltet eine Reihe solcher *Controller*. Der `cloud-controller-manager` dient zur Interaktion mit der API eines CSP. Als Beispiel einer solchen Interaktion lässt sich die automatische Bereitstellung von *Nodes* im Falle eines Ressourcenengpasses nennen (*Autoscaling*). Auf jedem *Node*, egal ob *Master* oder *Worker*, läuft `kubelet`, das die Ausführung der auf diesen *Node* zugewiesen *Pods* sicherstellt und als Schnittstelle zur *Control Plane* fungiert. Die Container eines *Pods* werden über eine entsprechende *Container Runtime* betrieben. Damit Kubernetes nicht an spezifische Laufzeitumgebungen gebunden ist, wurde eine einheitliche Schnittstelle, das *Container Runtime Interface*, geschaffen, welches von Container Laufzeitumgebungen, wie Docker oder containerd, implementiert wird. `kube-proxy` ist ein Netzwerk-Proxy, der die Kommunikation zwischen einzelnen Komponenten im Cluster, wie *Nodes* und *Pods*, ermöglicht [AD19, S. 33-35].

4.4. Modellierung und Implementierung

Für die in Abschnitt 2.3.2.1 und 2.3.2.2 vorgestellte Referenzarchitektur existieren bereits prototypische Implementierungen. Eine genaue Betrachtung der Schnittstellen sowie Beschreibungen über deren Kommunikationsverhalten und Aufgaben, führt zu folgendem Diagramm:[5]

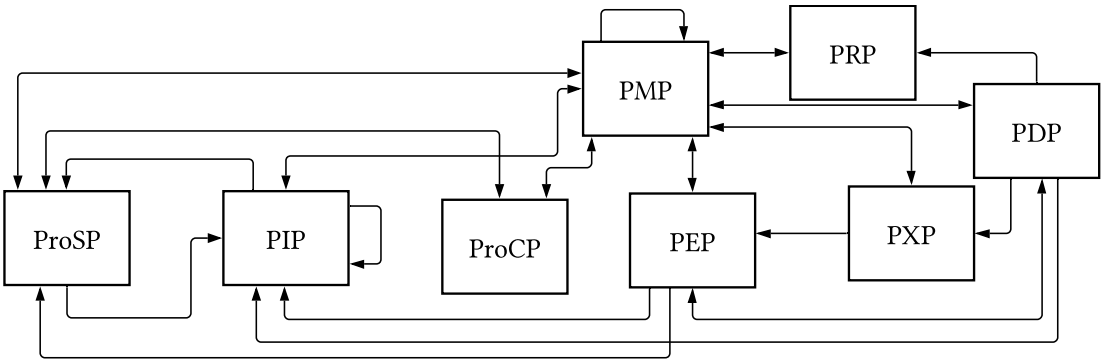

Abbildung 4.3.: Diagramm zur Veranschaulichung der Beziehungen zwischen den Komponenten der Referenzarchitektur

Hierbei fallen zwei Dinge auf: (i) Der PMP kennt jede Komponente. (ii) Die Beziehungen zwischen den Komponenten entsprechen nicht genau denen der Referenzarchitektur, sie sind wesentlich komplexer und teils bidirektional.

Punkt (ii) wird in der nachfolgenden Betrachtung weitestgehend ausgeklammert. Der Grund

5 Komponenten, die Pfeile auf sich selbst besitzen, stehen in Beziehungen zu denselben Komponenten, jedoch in anderen Domänen

ist, dass die Änderungsvorschläge für Punkt (i) unabhängig von Punkt (ii) durchgeführt werden können. Außerdem handelt es sich bei Punkt (ii) um ein Problem, bei dem man ohne entsprechendes Domänenwissen und detaillierten Anforderungen nur schwer gehaltvolle Aussagen treffen kann.

4.4.1. Modellierung

Die in diesem Abschnitt gezeigte Modellierung ist in Anlehnung an die *Cloud Native Maturity Matrix* in [RDG19, S. 63-84] entstanden (siehe hierzu Abbildung 1 im Anhang). Die Matrix beschreibt, wie unterschiedliche Prozesse, zum Beispiel das Wasserfallmodell oder Agile, Auswirkungen auf Architektur, Infrastruktur etc. haben können.

4.4.1.1. Stufe I: Die Grundlagen

In Stufe I wird die Modellierung der Problemstellung mithilfe von Kubernetes-eigenen Objekten angestrebt. Das Ergebnis ist in Abbildung 4.4 zu sehen. Der Kubernetes-Cluster besteht aus drei

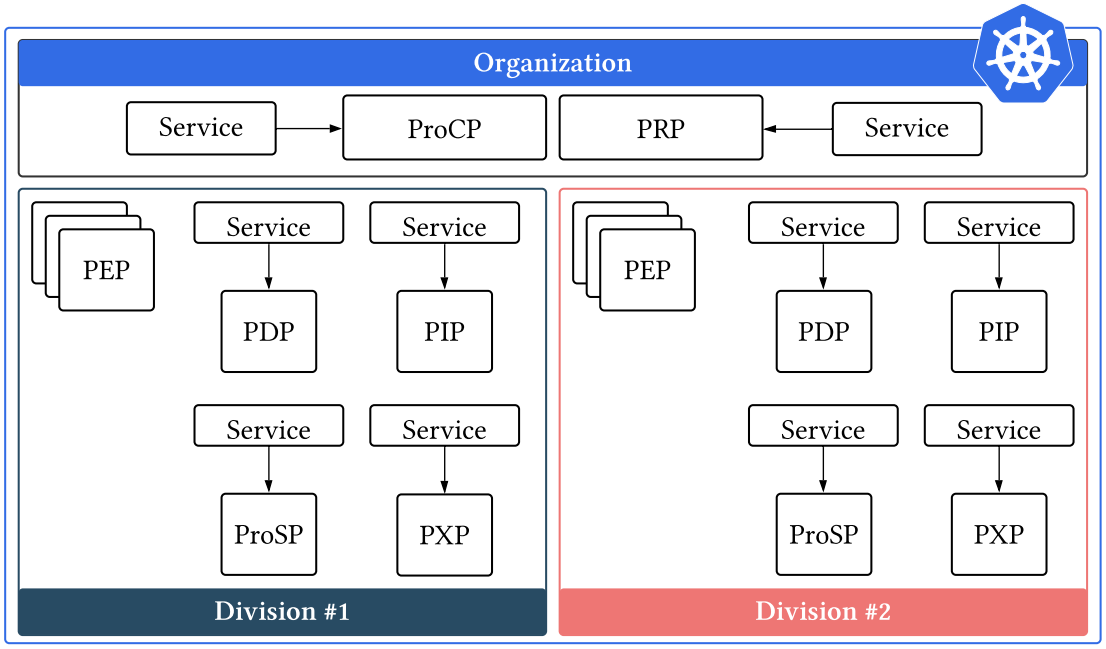

Abbildung 4.4.: Stufe I – Die Grundlage bildet Kubernetes

Namespaces, „Organization“, „Division #1“ und „Division #2“. Die Zuordnung der Komponenten orientiert sich an der Referenzarchitektur, in der alle Komponenten, außer der ProCP und PRP, immer Teil einer bestimmten Domäne sind, in unserem Fall der Domänen „Division #1“ und

„Division #2“. *Namespaces* sind in Kubernetes mit *Views* in Datenbanken vergleichbar. Das heißt, dass die in der Abbildung gezeigte Trennung vorerst rein logischer Natur ist und damit keinerlei Aussagen über die physische Platzierung der Komponenten macht.

Jeder der aus der Referenzarchitektur bekannten Komponenten stellt einen *Pod* beziehungsweise ein *Deployment* dar. Ein *Deployment* ist ein Kubernetes-eigenes Objekt, das die Verwaltung von *Pods* auf einer höheren Abstraktionsstufe ermöglicht. Die meisten dieser *Deployments* sind Teil eines *Service*, welche eine Menge von *Pods* gruppiert und ihnen einen festen Endpunkt vorgibt. Als Beispiel lässt sich ein *Service* mit dem Namen `pdp-service` im *Namespace* „Division #1“ anführen. Die Zuordnung, welche *Pods* zu welchem *Service* gehören, geschieht über Labels und entsprechende Selektoren. Kubernetes konfiguriert nun die `kube-proxy`-Komponente eines jeden *Nodes* und sorgt dafür, dass der *Service* über den Uniform Resource Locator (URL) `<service-name>.<namespace>` erreichbar ist, in unserem Fall also `pdp-service.division-1`.[6,7] Diesen Mechanismus nennt man auch *Service Discovery*. Kubernetes erlaubt *Service Discovery* über DNS, wie gerade beschrieben, als auch über Umgebungsvariablen sowie manuelles Setzen von *Endpoints*. Die Methode über DNS wird jedoch explizit empfohlen. Es ist auch möglich, Komponenten direkt anzusprechen, ohne dass diese Teil eines *Service* sind. Die Tatsache, dass *Service Discovery* in Kubernetes bereits integriert ist, macht den PMP unnötig. Ibryam & Huß beschreiben diesen Umstand wie folgt:

> „In the ‚Post Kubernetes Era‘, many of the nonfunctional responsibilities of distributed systems such as placement, health check, healing, and resource isolation are moving into the platform, and so is Service Discovery and load balancing.“ [IHD20, S. 100]

Die PEPs unterliegen der Annahme, Teil des Kubernetes-Cluster zu sein, theoretisch muss das nicht so sein. Man könnte sich mobile Geräte wie Laptops vorstellen, die nicht Teil des Clusters sind. Um diesen Geräten Zugriff zum Cluster zu geben, könnte ein Gateway etabliert werden (siehe hierzu Abbildung 2 im Anhang). An dieser Stelle sei noch einmal angemerkt, dass ein PEP auch aus einer Kombination von PEP, PIP und PXP bestehen könnte. Diese Kombination könnte ein *Pod* mit den entsprechend containerisierten Komponenten sein, welche über Inter Process Communication (IPC) miteinander kommunizieren.

4.4.1.2. Stufe II: Zustände auslagern

Einer der Kernprinzipien bei der Entwicklung von *Cloud Native*-Anwendungen ist *Operational Excellence*. In [SSJ19, S. 29-34] fallen hierzu die Schlüsselwörter *Automate Everything, Monitor*

6 Das Doppelkreuz stellt keinen gültigen Teil des Hostnamen dar und wurde deshalb weggelassen

7 <namespace> könnte weggelassen werden, sollte sich die Komponente im selben *Namespace* befinden

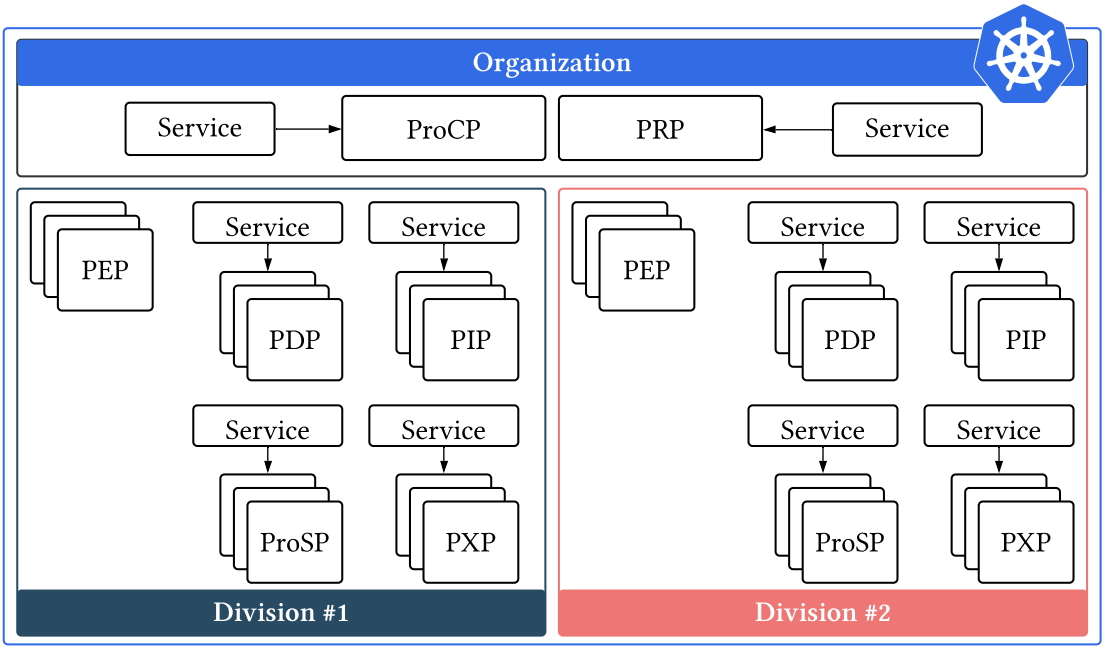

Abbildung 4.5.: Stufe II – Nutzung von *Replicas*

Everything, Document Everything und *Design for failure*. In dieser Stufe (siehe Abbildung 4.5), wird versucht, die Zustände der einzelnen Komponenten weitestgehend auszulagern. Durch Auslagerung der Zustände können mehrere Instanzen derselben Komponente gestartet und als ein gemeinsamer *Service* angesprochen werden. In Stufe I war die Architektur sehr anfällig gegenüber eines Ausfalls einzelner Komponenten. Ein Ausfall muss dabei nicht einmal auf einen Fehler zurückzuführen sein. Bereits die Aktualisierung einer Komponente hätte verheerende Folgen für den gesamten Betrieb. Bei einer enstprechend guten Verteilung der *Pods* auf möglichst unterschiedliche *Nodes*, könnten sogar einzelne *Nodes* ausfallen, ohne dabei den Betrieb ernsthaft zu gefährden. In diesem Kontext spricht man auch von einem System, welches eine *Zero Downtime* ermöglicht. Selbst bei Aktualisierung einzelner Komponenten ist es mit Kubernetes möglich, den Betrieb weiterhin aufrechtzuhalten. Im Anhang (siehe Abbildung 3) befindet sich noch einmal die Abbildung 4.5 mit entsprechendem Gateway für externe PEPs.

4.4.1.3. Stufe III und IV: Weitere Ergänzungen

Die zwei nächsten Stufen greifen weitere Verbesserungsvorschläge auf. Stufe III (siehe Abbildung 4.6) stellt die Frage, ob eine Aufteilung des Kubernetes-Cluster nach Organisationseinheiten sinnvoll erscheint, da die Zustände nun ausgelagert sind. Hierbei lässt sich argumentieren,

dass die Auftrennung in Organisationseinheiten erst bei einer entsprechenden Größe, wie Mutterkonzern und Tochterkonzern sinnvoll ist, da in diesem Fall auch eine entsprechend physisch aufgeteilte Infrastruktur vorhanden sein dürfte. Sofern Regulierungen es nicht anders vorschreiben, könnte darauf verzichtet werden. Der Autor Bier [Bie17] hatte die hierarchische Architektur etabliert, um Daten möglichst lokal zu halten und verwalten zu können. In der Praxis dürften Unternehmen jedoch daran interessiert sein, Rechenkapazität zu zentralisieren, um dadurch Kosten einsparen zu können. Ein hierarchisches System sorgt in diesem Fall nur für zusätzliche Komplexität.

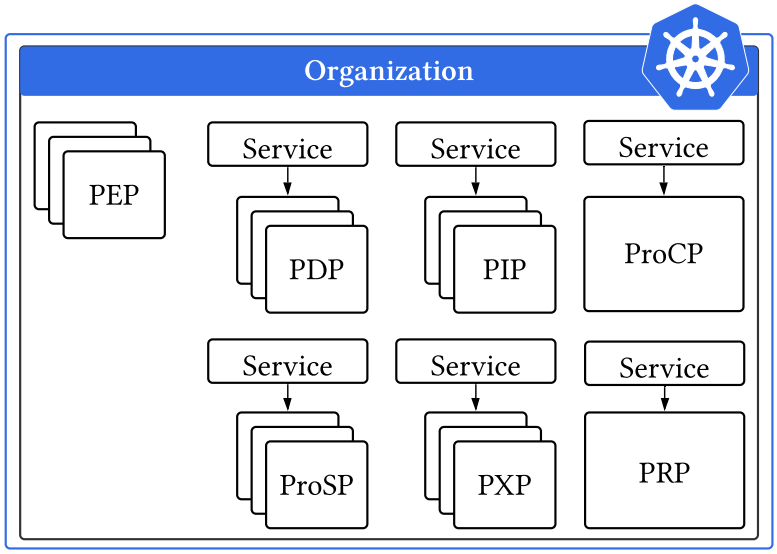

Abbildung 4.6.: Stufe III – Entfernung der Namespaces

Stufe IV sieht die Einführung eines *Service Meshes* vor. Ein *Service Mesh* ergänzt den Kubernetes-Cluster um zusätzliche Funktionalität auf *Service*-Ebene, wie *Traffic Management, Failure Handling, Security, Tracing* und *Monitoring* [SSJ19, S. 59-72]. Das Thema *Service Mesh* ist jedoch sehr umfrangreich und soll an dieser Stelle nicht weiter vertieft werden.

4.4.1.4. Stufe V: Eventbasiert und Serverless

In Stufe V wird weitestgehend von der physischen Infrastruktur abstrahiert. Die zentrale Komponente dieser Architektur ist eine Event-Queue, in Abbildung 4.7 als „Pub/Sub“ bezeichnet, die den Start von *serverless* betriebenen Containern anstößt. Die Event-Queue funktioniert nach dem *Publish-Subscribe Pattern*. Komponenten schicken Events an Kanäle, *Topics* genannt. Diese Events werden daraufhin von anderen Komponenten bearbeitet, die an spezifische *Topics* vorher *subscribed* haben. In unserem Fall löst ein solches Event jedoch automatisch den Start

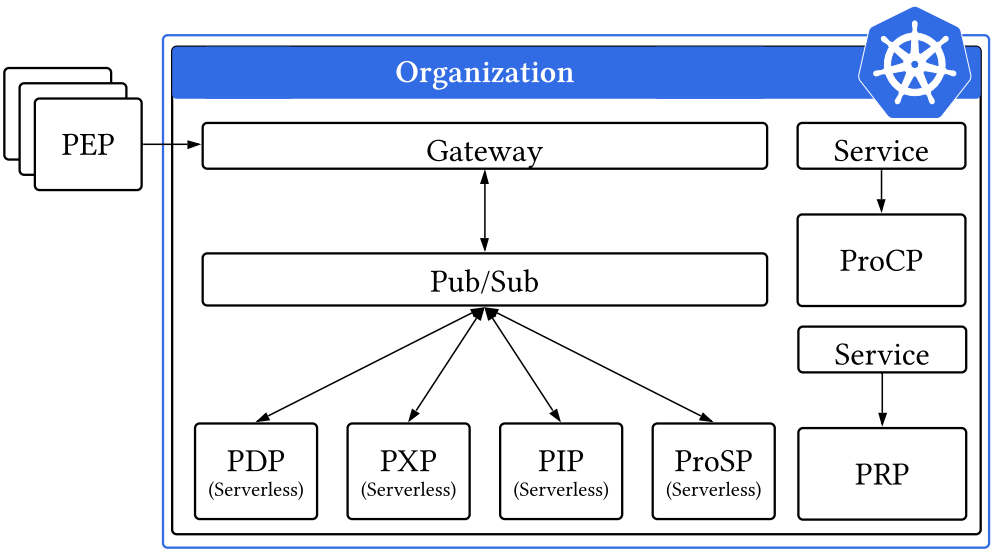

Abbildung 4.7.: Stufe V – Umsetzung einer eventbasierten *Serverless*-Architektur

eines Containers beziehungsweise einer Funktion aus. Diese nimmt das Event aus der Event-Queue, bearbeitet das Event, generiert hierbei eventuell weitere Events und beendet sich wieder. Die Kernmerkmale dieser Architektur sind eine sehr hohe Skalierbarkeit, Fehlertoleranz sowie effiziente Nutzung der vorhandenen Ressourcen. Anstelle von Prozessen, die über eine lange Zeit laufen müssen, kann hierbei die Ausführung bedarfsgerecht erfolgen, das Ergebnis ist maximale Elastizität. Die Event-Queue selbst kann ein verteiltes System sein, wie zum Beispiel Apache Kafka.[8]

4.4.2. Implementierung

Die Implementierung der Stufen I-III wurde mithilfe von Kubernetes und *Dummy*-Komponenten realisiert.[9] Jeder der Komponenten wurde in Python programmiert und steht unter der *MIT-License*. Als Web-Framework wurde FastAPI verwendet.[10] Mithilfe der GitLab CI/CD wird bei jedem *Push* automatisiert die Einhaltung des *Code Styles* überprüft,[11] Tests ausgeführt (außer beim PEP) und das entsprechende Docker Image gebaut und auf DockerHub hochgeladen.[12] Aus Gründen der Übersicht wurden alle Images auf dasselbe Docker-Repository hochgeladen

8 `https://kafka.apache.org` (besucht am 25. 11. 2020)
9 Das zugehörige Repository ist hier zu finden: `https://gitlab.com/bachelorarbeit1` (besucht am 25. 11. 2020)
10 `https://fastapi.tiangolo.com` (besucht am 26. 11. 2020)
11 Verwendet wurde der black *Code Style* `https://github.com/psf/black` (besucht am 26. 11. 2020)
12 `https://hub.docker.com/repository/docker/nschuler/thesis` (besucht am 25. 11. 2020)

Endpunkt	Beschreibung
/event	Zum Senden eines Events an die Komponente †
/info	Informationen über die Komponente †
/ws	Endpunkt, mit dem eine Websocket-Verbindung aufgebaut werden kann
/docs	Aufzeigen der OpenAPI-Spezifikation mithilfe der Swagger UI
/redoc	Aufzeigen der OpenAPI-Spezifikation mithilfe von ReDoc

Tabelle 4.1.: Zur Verfügung stehende Endpunkte der *Dummy*-Komponenten

und lediglich entsprechend getagged. Damit die Container ordnungsgemäß gestartet werden können, müssen mehrere Umgebungsvariablen beim Start des Containers übergeben werden. Dies geschieht automatisiert durch Kubernetes. Ein Großteil der Variablen wird durch die *Downward API* von Kubernetes injiziert, welche die Umgebungsvariablen dynamisch erzeugt. Als Beispiel sei hierfür die Umgebungsvariable mit den Namen des *Nodes* genannt, auf dem der Container beziehungsweise *Pod* ausgeführt wird.

Die PEPs werden als *DaemonSet* gestartet. Dies hat zur Folge, dass auf jedem Node ein PEP platziert wird. Die PEPs versuchen daraufhin, über den Websocket-Endpunkt `/ws` eine Verbindung zu einem PDP und PIP aufzubauen. Der genaue Kommunikationsablauf kann aus Abbildung 4.8 entnommen werden. Die `par` Box stellt eine parallele Ausführung dar. Die *Dummy*-Komponten verwenden hierfür `asyncio` welches einen asynchronen Kommunikationsablauf ermöglicht. Der Grund für die Nutzung von Websockets war das Etablieren einer bidirektionalen Verbindung. Nachrichten können ohne erneuten Verbindungsaufbau vom PEP empfangen und gesendet werden. Die anderen Komponenten starten einen Webserver, welcher die in Tabelle 4.1 dargestellten Endpunkte bereitstellt. Jedoch stellen nur PDP, PIP und ProSP den Endpunkt `/ws` zur Verfügung. Zur Visualisierung sind im Anhang OpenAPI-Spezifikation sowie dazugehörige JSON-Objekte mit beispielhaften Werten angegeben.

Eine der Anforderungen war es, genauer bestimmen zu können, wo *Pods scheduled* werden. Wie bereits angesprochen handelt es sich bei der Unterteilung in *Namespaces* lediglich um eine rein logische, welche keinen Einfluss auf die Platzierung der *Pods* auf *Nodes* hat. Es existieren zwar Plugins, mit denen dies möglich ist (siehe hierzu *PodNodeSelector*), aber diese sind standardmäßig deaktiviert und bei *Managed Cluster*, wie Google's GKE, nicht aktivierbar. Mit den in Kubernetes standardmäßig vorhandenen Mitteln lassen sich die in Abbildung 4.9 gezeigten Attribute selektieren. Dazu gehören *Pools* von *Nodes*, Namen von *Nodes* sowie Labels, mit denen die *Nodes* versehen wurden.[13] Die Selektion über den Namen eines *Nodes* wird jedoch offiziell nicht empfohlen, weshalb die beiden anderen Möglichkeiten benutzt

† Siehe hierzu die JSON-Objekte und OpenAPI-Spezifikation im Anhang

13 Kubernetes erlaubt die Zuweisung von Labels auf fast alle Objekte

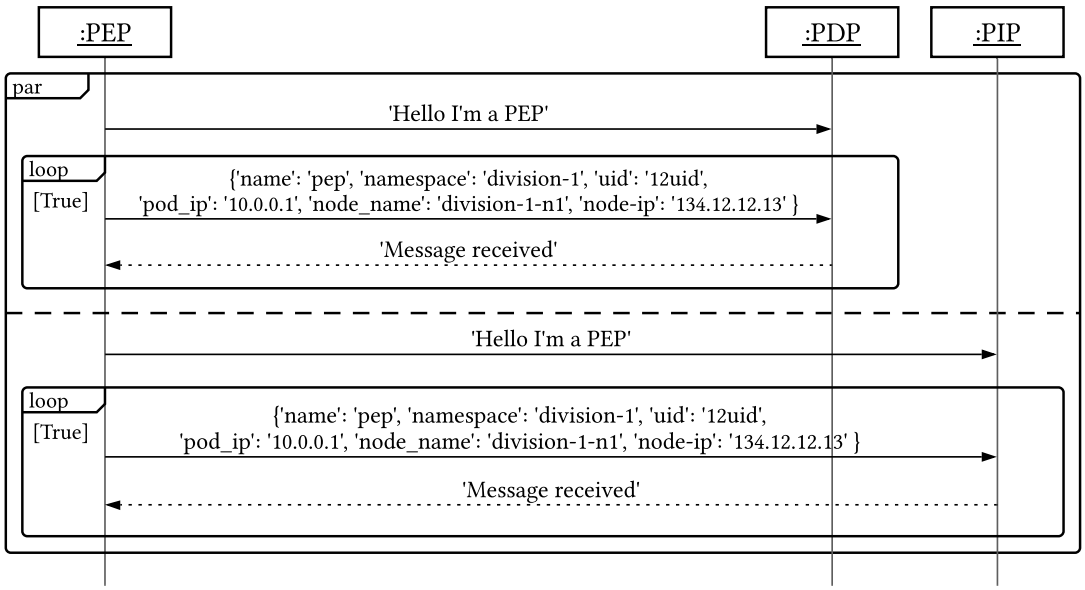

Abbildung 4.8.: Sequenzdiagramm zur Kommunikation von PEP, PDP und PIP

wurden. Die Selektion geschieht über den `nodeSelector`. Dies wurde sowohl lokal mithilfe von minikube,[14] als auch in der Google Cloud mit GKE getestet.

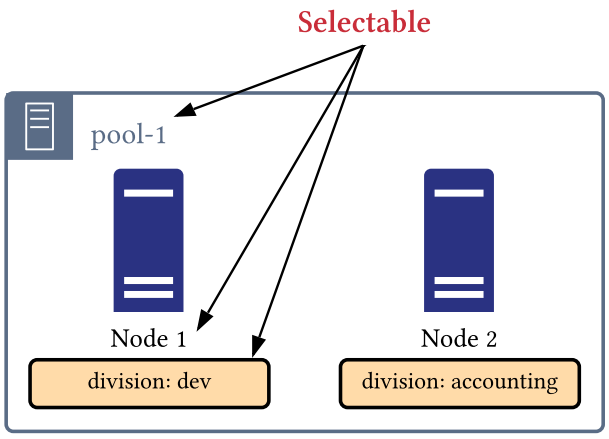

Abbildung 4.9.: Selektierbare Eigenschaften in Kubernetes

Hierbei zeigte sich, dass die manuelle Erstellung der Kubernetes-Manifeste für komplexere Szenarien, wie mehrere Organisationseinheiten, mit sehr viel Redundanz einhergeht. Diese Redundanz ist potentiell anfällig für Fehler, sollten Änderungen an manchen Stellen

14 `https://minikube.sigs.k8s.io` (besucht am 26. 11. 2020)

```yaml
divisions:
  - name: "Research and Development"
    namespace: "research"
    description: "Department of development. Person in charge is Sundar Pichai."
    [...]
  - name: "Accounting"
    namespace: "accounting"
    description: "Department of accounting. Person in charge is Ruth Porat."
    [...]
  - name: "Marketing"
    namespace: "marketing"
    description: "Department of marketing. Person in charge is Tim Cook."
    [...]
```

Listing 1: Beispiel einer Konfigurationsbeschreibung mit YAML

vergessen werden. Um dem entgegenzuwirken und den Prozess zur Erstellung der Kubernetes-Manifeste zu automatisieren, existiert Helm.[15] Helm bezeichnet sich selbst als Paketmanager für Kubernetes. Die Grundidee hinter Helm ist, mithilfe von Templates die Kubernetes Manifeste zu beschreiben und die konkrete Konfiguration an zentralen Stellen wie YAML-Dateien auszulagern. Die Templates können dann gerendert und die daraus erzeugten Manifeste im Kubernetes-Cluster *deployed* werden. Das templating wird mit Go-Templates und weiteren ergänzenden Funktionen realisiert. Templating ist ein sehr mächtiges Werkzeug, da hiermit Schleifen und weitere Programmierkonstrukte für die Erstellung der Manifeste nutzbar sind. Die Beschreibung der Konfiguration geschieht bei Helm im YAML-Format. YAML ist eine Obermenge von JSON und ideal zur Konfigurationsbeschreibung geeignet, unter anderem deswegen, da Konfigurationen anhand von Schemata validierbar sind. Ein Beispiel für eine solche Konfigurationsbeschreibung ist in Listing 1 zu sehen. Darin wird eine Organisation mit drei Organisationseinheiten beschrieben. Mithilfe vorbereiteter Templates können daraus konkrete Kubernetes-Manifeste erzeugt werden, die dann zur Bereitstellung der Nutzungskontroll- und Provenance Tracking-Infrastruktur genutzt werden können. Die Templates für obiges Beispiel sind im Ordner `helm-package/ucon-prov/templates` des `kubernetes`-Repository in GitLab zu finden. Ausschnitte des Quellcodes befinden sich am Ende des Anhangs.

15 `https://helm.sh` (besucht am 25. 11. 2020)

5. Evaluation und Diskussion

In diesem Kapitel wird besprochen, ob und wie die zu Beginn von Kapitel 4 gesteckten Ziele erreicht wurden. Im Anschluss erfolgt eine Diskussion über weitere Verbesserungspotenziale vor allem im Hinblick auf das Thema Sicherheit.

5.1. Evaluation

Das erste Ziel bezog sich darauf, die Kommunikation zwischen den Komponenten zu vereinfachen. Dieses Ziel wurde in Teilen erfüllt. Durch die Entfernung des PMP, welcher für die *Service Discovery* zuständig war, konnte die Verantwortlichkeit aus der Referenzarchitektur in Kubernetes verschoben werden. In Hinblick auf die vorherige Kommunikation werden hierdurch hauptsächlich *Lookup*-Anfragen eingespart. Bis auf Stufe V und der Tatsache, dass der PEP eine Websocket-Verbindung zum PDP und PIP aufbaut, blieb die Kommunikation unangetastet. Die Websocket-Verbindung ist eine bessere Repräsentation des Zustandsautomaten, der ständig eine Neubewertung der Nutzungserlaubnis vornimmt. Die Idee hierbei ist, dass der Zustand zumindest teilweise an die Websocket-Verbindung gekoppelt werden kann. In dieser Hinsicht ist es praktisch, eine beständige, bidirektionale Verbindung aufzubauen. Stufe V wurde zwar nicht implementiert, jedoch ist stark davon auszugehen, dass die Kommunikation sich wesentlich vereinfachen würde. Auch käme man weitestgehend weg von einem synchronen *Request/Response Model*, hin zu einem asynchronen Modell, das sich weitestgehend selbst orchestrieren kann.

Das zweite Ziel verfolgte die Verbesserung operativer Aspekte, insbesondere des Konfigurationsmanagements. Mussten die Komponenten vorher manuell bereitgestellt und konfiguriert werden, so kann dies nun automatisiert erfolgen. Konkret wurde die Bereitstellung der Komponenten wesentlich vereinfacht. Hierfür muss lediglich ein Kubernetes-Manifest erstellt und Kubernetes übergeben werden. Kubernetes sorgt danach automatisch für die Platzierung der *Pods*, Konnektivität mit dem Cluster-Netzwerk, das Setzen von Umgebungsvariablen, *Secret*-Management, Anbindung von Speicher, die Einhaltung und Berücksichtigung von Ressourcen-Quotas, Überwachung des Systemzustandes (*Health-Checking*), sowohl von Containern als auch *Nodes*, und mehr. Auch das *Debugging* ist wesentlich vereinfacht, da, sofern die enstpre-

chenden Rechte vorhanden sind, eine Betrachtung der Logs *remote* über die Kommandozeile möglich ist. All diese Funktionalität hatte im Grunde nichts mit den Komponenten der Referenzarchitektur zu tun und sollte daher nicht Teil der eigentlichen Architektur sein. Durch Kubernetes wurde eine klarere Trennung der Verantwortlichkeiten zwischen Anwendungs- und Infrastrukturebene erreicht.

Ferner wurde auch das letzte Ziel erfüllt, bei dem es um die automatisierte Bereitstellung einer Nutzungskontroll- sowie Provenance sammelnden Infrastruktur ging. Für den weiteren Ausbau dieser Idee müssten die Templates sowie die Konfigurationsbeschreibung möglichst generisch gehalten werden, sodass diese auf möglichst viele Organisationsformen anwendbar ist. Die Konfigurationsbeschreibung sollte durch ein JSON-Schema validierbar gemacht werden, um deren Benutzung zu erleichtern. Jedoch darf man nicht vergessen, dass diese Aussage unter der Annahme eines bereits existierenden und konfigurierten Kubernetes-Clusters stand. Wird diese Annahme weggelassen, so liegt eine deutliche Minderung der automatisierten Bereitstellung vor.

5.2. Diskussion

Die Entscheidung, auf Kubernetes aufzubauen, lässt sich anhand der Servicemodelle aus Abschnitt 3.2 erklären. Kubernetes sitzt als Container-Orchestrator zwischen IaaS und SaaS. Eine IaaS-Lösung hätte zwar weitaus mehr Flexibilität zur Folge gehabt, aber die Komplexität sowie der Verwaltungsaufwand wären um ein Vielfaches höher gewesen. Bei einer PaaS-Lösung wäre genau der umgekehrte Fall eingetreten. Kubernetes bot damit einen optimalen Kompromiss zwischen den beiden Servicemodellen. Darüber hinaus gibt es nicht viele Systeme, die mit Kubernetes vergleichbar wären.

Der nächste Punkt greift auf, warum versucht wurde, mit der hierarchischen Systemarchitektur der Referenzarchitektur zu brechen. Einer der Gründe war Abstraktion. Während mit Cloud-Technologien versucht wird, weitestgehend von der darunterliegenden Infrastruktur zu abstrahieren, tat die Referenzarchitektur genau das Gegenteil. Die Folge war eine systeminduzierte hohe Kopplung zwischen Anwendung und Infrastruktur. Zwar lässt sich dies mit Technologie beherrschbar machen, die korrekte Lösung wäre jedoch, dem Problem auf der richtigen Abstraktionsebene zu begegnen und die Verantwortlichkeiten sauber zu trennen. Der zweite Grund ist die erhöhte Komplexität, die eigentlich eine Begleiterscheinung des ersten Grundes ist.

Die saubere Trennung der Verantwortlichkeiten betrifft auch Eigenschaften der Sicherheit, die eindeutig der Infrastruktur zuzuordnen sind. Hierbei stellt sich die Frage, welche Sicherheitseigenschaften durch Kubernetes bereits gegeben sind und welche man zusätzlich ergänzen

könnte. Im bestmöglichen Fall sollten so viele Sicherheitseigenschaften wie möglich in die Infrastrukturebene integriert werden, sodass die Anwendungsebene diese nur noch „benutzen“ muss. Eine der sowohl für Nutzungskontrolle als auch für Provenance Tracking notwendigen Sicherheitseigenschaften ist gegenseitiges Vertrauen. Hierzu wurden in Abschnitt 3.4.3.1 mehrere Möglichkeiten zur Etablierung eines *Trusted Subsystem* in der Cloud besprochen. Dabei wurde festgestellt, dass bereits Erweiterungen für *Confidential Computing* in Kubernetes existieren.[1] Soweit dem Autor der vorliegenden Arbeit bekannt, muss die *Remote Attestation* manuell angestoßen werden. Kubernetes bietet jedoch eine Vielzahl an Erweiterungsmöglichkeiten, mit denen dieser Prozess automatisiert werden könnte.[2] Ein Großteil der Erweiterungen muss dabei nicht einmal direkt im Cluster laufen, sondern kann auch außerhalb dessen aufgerufen werden. Auch eine Erweiterung der deklarativen API von Kubernetes ist möglich. Zum Beispiel könnte man sich Selektoren vorstellen, die die Platzierung von *Pods* auf *Nodes* anhand von Sicherheitseigenschaften ermöglichen, wie eben der Eigenschaft einen Prozessor zu besitzen, der *Confidential Computing* unterstützt. Neben solchen *Device Plugins* sind auch *Network-* und *Storage Plugins* möglich, zum Beipiel unterstützt Kubernetes seit nun mehreren Versionen die automatische Verschlüsselung von Daten *at rest*. Selbst der `scheduler` von Kubernetes kann erweitert oder sogar selbst geschrieben werden. Weitere Möglichkeiten der Erweiterung sind *Authentication und Authorisation Webhooks*, sollte das Kubernetes-eigene RBAC-System nicht ausreichen. Oftmals genügt es schon, die Kubernetes-API um Ressourcen, sogenannte Custom Resource Definitions (CRDs), zu erweitern. Der Zustand dieser CRDs kann dann von eigens geschriebenen Controllern überwacht werden, welche gegebenenfalls Anpassungen vornehmen, um den Soll-Zustand wiederherzustellen (*Operator Model*). Eine weitere Form der Erweiterung sind *Admission Controller*, welche Anfragen an die Kubernetes-API abfangen, um daran Manipulationen vorzunehmen oder diese ganz zu blockieren. Auch die Integration von Key Management Services (KMSs) könnte durch entsprechende Erweiterungen realisiert werden. Darüber hinaus können über das *Container Runtime Interface* Containter-Laufzeitumgebungen genutzt werden, die als vergleichsweise „sicher“ gelten.

Die beschriebene Flexibilität geht jedoch mit einer gewissen Komplexität einher. Diese beginnt bereits beim Aufsetzen eines Clusters. Hierfür wird empfohlen, dass jeder *Node* mit dem *Public Root Certificate* des Clusters und dem *Client Certificate* des *Nodes* aufgesetzt wird. Kubernetes kann seit Version 1.8 beide Zertifikate automatisch rotieren. Initial ist dies jedoch wichtig, damit direkt eine sichere Verbindung zum Cluster aufgebaut werden kann. Selbst wenn das Aufsetzen eines Kubernetes-Clusters noch vergleichsweise einfach sein könnte, so muss

1 Intel realisiert diese zum Beispiel als *Device Plugin* `https://github.com/intel/intel-device-plugins-for-kubernetes` (besucht am 26. 11. 2020)

2 Siehe hierzu `https://kubernetes.io/docs/concepts/extend-kubernetes` (besucht am 26. 11. 2020)

dieser, wie jedes andere System auch, von entsprechend qualifiziertem Personal betreut werden. Aufgrund der deklarativen Form der Kubernetes-API besitzt Kubernetes die Fähigkeit, sich selbst zu heilen. Sollte zum Beispiel ein *Pod* ausfallen, so startet Kubernetes diesen automatisch neu, um den Soll-Zustand wiederherzustellen. Darüber hinaus bietet Kubernetes Möglichkeiten einer feingranularen Label-Selektion, Prioritätsklassen für *Pods*, das Vorgeben von Werten für AppArmor, SELinux sowie *Linux Capabilities* und Möglichkeiten zum Setzen von *Ingress* und *Egress* Policies.

Anstelle der Websocket-Verbindung hätte auch gRPC verwendet werden können.[3] Dieses Protokoll ermöglicht auch das Aufbauen einer bidirektionalen Verbindung, allerdings hat es den Vorteil, typisiert zu sein und auf HTTP/2 aufzubauen. Auch sind Authentifizierungsmechanismen bereits in das Protokoll integriert.

Organisationsübergreifende Nutzungskontrolle und Provenance Tracking könnte durch eine Föderation von Kubernetes-Clustern realisiert werden. Die Cluster müssten hierfür *Ingress* Routen und *Ingress Controller* bereitstellen, die Anfragen von außerhalb an die entsprechenden *Services* weiterleiten. Schwieriger ist die Etablierung des gegenseitigen Vertrauens zwischen diesen Clustern. Dem Autor der vorliegenden Arbeit wäre nur eine Lösung über eine *Trusted Third Party* bekannt. Sollten sich die zwei Cluster beim selben CSP befinden, so könnte dies der CSP sein, davon ist jedoch im Allgemeinen nicht auszugehen. Eine weitere Möglichkeit wäre die Erweiterung von Kubernetes, so wie oben bereits angesprochen. Die hierfür zuständigen Controller könnten außerhalb des Clusters bei einer *Trusted Third Party* liegen. Als Szenario könnte man sich vorstellen, dass jeder Cluster ein KMS bereitstellt. Die Daten könnten daraufhin verschlüsselt von einem Cluster zu einem anderen übertragen werden, wobei der Schlüssel zunächst im KMS des jeweiligen Clusters verbleibt. Der Cluster muss nun überprüfen können, ob es sicher ist den Schlüssel zu übertragen. Diese Sicherheit kann er durch *Confidential Computing* erlangen. Hierfür muss die Anwendung, welche den Schlüssel entgegennehmen soll, attestiert werden. Die Anwendung und dessen Speicherbereich im Hauptspeicher werden durch den Prozessor geschützt, sodass von außen kein Zugriff auf den Schlüssel möglich ist.

Weitere Forschung sollte zunächst der in dieser Arbeit beschriebenen Fall der Etablierung des gegenseitigen Vertrauens mithilfe von Kubernetes innerhalb einer Organisation untersuchen. Die daraus resultierenden Erkenntnisse können dann für den organisationsübergreifenden Fall evaluiert und gegebenenfalls genutzt werden.

3 `https://grpc.io` (besucht am 26. 11. 2020)

6. Fazit und Ausblick

Im Folgenden werden die in der Arbeit gesammelten Erkentnisse zusammengefasst. Anschließend erfolgt ein Ausblick auf zukünftige Arbeiten.

6.1. Fazit

Daten haben sich in den letzten Jahrzehnten zu einem der wertvollsten Rohstoffe entwickelt. Umso wichtiger erscheint es, die Nutzung von Daten möglichst transparent und kontrollierbar zu gestalten. Zu diesem Zweck wurde auf die Themen Nutzungskontrolle und Provenance Tracking eingegangen. Zusammengefasst ermöglicht Nutzungskontrolle Zugriffskontrolle auf Daten, auch nachdem diese bereits verbreitet wurden, während Provenance Tracking darauf abzielt, die Herkunft von Daten nachvollziehbar zu machen. Neben den theoretischen Grundlagen wurde in beiden Fällen auf Überschneidungen mit dem Themengebiet *Cloud Computing* eingegangen. Hierbei zeigte sich, dass die Etablierung des gegenseitigen Vertrauens – notwendig sowohl für Nutzungskontrolle als auch Provenance Tracking – zu einer der schwierigen offenen Forschungsfragen zählt. Ferner wurde durch die Betrachtung des IDS eine Referenzarchitektur zur Nutzungskontrolle und des Provenance Trackings näher beleuchtet.

Im Anschluss dazu wurde ein grundlegendes Verständnis für den Begriff Cloud geschaffen. Vertieft wurde dieses Verständnis durch eine genauere Betrachtung der Aspekte Sicherheit und Recht. Hierbei wurde festgestellt, dass sich die Sicherheit in der Cloud nur bedingt zur Sicherheit in „traditionellen“ IT-Systemen abgrenzt. Zertifizierungen helfen, zusätzliches Vertrauen zu schaffen und Rahmenbedingungen zu klären. Selbst datenschutzrechtlich ist die Cloud nicht schlechter gestellt als jedes andere IT-System. Nach der DSGVO ist jedoch die Ansässigkeit der großen CSPs in den USA problematisch. Darüber hinaus wurde aufgezeigt, dass die Cloud über ausreichend Schutzmaßnahmen verfügt, um sowohl Daten *in motion* und *at rest*, als auch Daten *in use* zu schützen. Auch wurde auf eine Reihe möglicher Bedrohungen und Besonderheiten im Kontext des *Cloud Computing* eingegangen, wie Denial of Service (DoS)-Attacken und die Sicherheit von VMs und der *Cloud Console*.

Diese Erkenntnisse wurden dann zur Ausbringung einer prototypischen Nutzungskontroll- sowie Provenance sammelnden Infrastruktur mithilfe von Kubernetes und *Dummy*-Komponen-

ten der Referenzarchitektur genutzt. Die Modellierung wurde auf Basis eines *Cloud Native*-Ansatzes anhand fünf unterschiedlicher Reifegrade vorgenommen. In der Implementierung der Stufen I-III zeigte sich eine wesentliche Verbesserung operativer Aspekte. So konnte sowohl die Kommunikation, als auch das Konfigurationsmanagement und die Administration der Komponenten vereinfacht werden. Darüber hinaus wurde eine Möglichkeit beschrieben, wie die Infrastruktur automatisiert bereitgestellt werden kann. In der abschließenden Diskussion wurde dargelegt, wie wichtig eine saubere Trennung der Verantwortlichkeiten zwischen Anwendung und Infrastruktur ist. Ferner wurden Limitierungen, Erweiterungen und Verbesserungsmöglichkeiten, auch in Bezug auf eine organisationsübergreifende Nutzungskontrolle und Provenance Tracking, aufgezeigt.

6.2. Ausblick

Die in dieser Arbeit gewonnenen Erkenntnisse sind aufgrund ihrer Verwendung von *Dummy*-Komponenten zwar nur prototypisch zu betrachten, dennoch wurde damit der Grundstein für weitere Forschung gelegt. Allem voran wurden die Stufen IV-V nicht implementiert und in der Folge nicht evaluiert. Dabei bieten sowohl das *Service Mesh* aus Stufe IV als auch die eventbasierte *Serverless*-Architektur aus Stufe V weitere Verbesserungspotenziale. So könnte mithilfe des *Service Mesh* Nutzungskontrolle auf Ebene von *Services* untersucht werden und mithilfe von *Distributed Tracing*[1] könnte das Provenance Tracking weiter ausgebaut werden. Darüber hinaus könnte sich Stufe V als ideales Architekturmuster für *Confidential Computing* herausstellen. An dieser Stelle sei angemerkt, dass das in dieser Arbeit vorgestellte Reifegradmodell keine Aussagen über die letztendlich „beste“ Umsetzung macht.

Auch wurden viele der Herausforderungen, bezogen auf Provenance Tracking im Kontext von *Cloud Computing* aus Abschnitt 2.2.2, nicht beantwortet. An dieser Stelle könnte weitere Forschung ansetzen.

Ein Thema, das sich bereits durch die gesamte Arbeit zieht, ist die Etablierung des gegenseitigen Vertrauens. Hierfür wurden Ansätze des *Trusted Computing* als auch des *Confidential Computing* angesprochen. Die Integration sowie Untersuchung der Praxistauglichkeit in Kubernetes steht jedoch noch aus. Dabei könnten dies Schlüsseltechnologien für den organisationsübergreifenden Fall der Nutzungskontrolle und des Provenance Trackings sein. Hierzu merkt Gartner an, dass es noch fünf bis zehn Jahre dauern soll, bis Technologien wie *Confidential Computing* massentauglich werden [Gartner20D]. Auch gilt es zu untersuchen, wie viele Sicherheitseigenschaften in die Infrastrukturebene – in diesem Fall Kubernetes – integriert werden

1 Zum Beispiel mit Jaeger `https://www.jaegertracing.io` (besucht am 08. 12. 2020)

können. Abschnitt 5.2 beschreibt hierzu eine Vielzahl von Erweiterungspunkten und Verbesserungen. Ferner kann die automatisierte Bereitstellung sowie die Konfigurationsbeschreibung mithilfe der Templates weiter ausgebaut werden.

Literatur

[A4Cloud] Brian (QMUL) Dziminski et al. *D:C-2.1 Report detailing conceptual framework*. Hrsg. von Massimo (HP) Felici und Siani (HP) Pearson. 2014. URL: http://cloudaccountability.eu/sites/default/files/D32.1%20Conceptual%20Framework.pdf (besucht am 09. 10. 2020).

[AD19] J. Arundel und J. Domingus. *Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud*. O'Reilly Media, 2019. ISBN: 9781492040712.

[Al+18] Y. Al-Dhuraibi et al. „Elasticity in Cloud Computing: State of the Art and Research Challenges". In: *IEEE Transactions on Services Computing* 11.2 (2018), S. 430–447. DOI: 10.1109/TSC.2017.2711009.

[Ali+10] Tamleek Ali et al. „On Usage Control of Multimedia Content in and through Cloud Computing Paradigm". In: *2010 5th International Conference on Future Information Technology*. IEEE. 2010, S. 1–5. DOI: 10.1109/FUTURETECH.2010.5482751.

[Ana+14] G. F. Anastasi et al. „Usage Control in Cloud Federations". In: *2014 IEEE International Conference on Cloud Engineering*. IEEE. 2014, S. 141–146. DOI: 10.1109/IC2E.2014.58.

[Bat+13] Adam Bates et al. „Towards Secure Provenance-based Access Control in Cloud Environments". In: *Proceedings of the third ACM conference on Data and application security and privacy*. 2013, S. 277–284. DOI: 10.1145/2435349.2435389.

[BBH19] B. Burns, J. Beda und K. Hightower. *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. O'Reilly Media, Incorporated, 2019. ISBN: 9781492046530.

[Bec03] Eberhard Becker, Hrsg. *Digital Rights Management : Technological, Economic, Legal and Political Aspects*. Lecture notes in computer science ; 2770. Berlin: Springer, 2003, S. 3–4. ISBN: 3540404651. DOI: 10.1007/b12637.

[Bec20] Markus Beckedahl. *EuGH zum Privacy Shield - Schuld ist das System der Massenüberwachung*. Juli 2020. URL: https://netzpolitik.org/2020/eugh-zum-privacy-shield-schuld-ist-das-system-der-massenueberwachung (besucht am 29. 10. 2020).

[Bha19] Kiran Bhageshpur. *Data Is The New Oil – And That's A Good Thing*. Igneous. Nov. 2019. URL: https://www.forbes.com/sites/forbestechcouncil/2019/11/15/data-is-the-new-oil-and-thats-a-good-thing (besucht am 04. 12. 2020).

[Bie17] Philipp Christoph Sebastian Bier. *Umsetzung des datenschutzrechtlichen Auskunftsanspruchs auf Grundlage von Usage-Control und Data-Provenance-Technologien*. 2017. URL: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20171221-1356835-1-5.

[Bra20] Mary Branscombe. *Microsoft Azure Brings Confidential Computing to Kubernetes*. Sep. 2020. URL: https://thenewstack.io/microsoft-azure-brings-confidential-computing-to-kubernetes (besucht am 06. 11. 2020).

[Bre00] Eric A. Brewer. „Towards Robust Distributed Systems". In: *PODC*. Bd. 7. 10.1145. Portland, Oregon, USA, 2000. DOI: 10.1145/343477.343502.

[Bri19] Volker Briegleb. *Cloud-Hochzeit: IBM schließt Übernahme von Red Hat ab*. heise online. Juli 2019. URL: https://heise.de/-4466564 (besucht am 04. 12. 2020).

[BSI-CC] *Eckpunktepapier Sicherheitsempfehlungen für Cloud Computing Anbieter*. Bundesamt für Sicherheit in der Informationstechnik (BSI), Feb. 2012. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/Eckpunktepapier-Sicherheitsempfehlungen-CloudComputing-Anbieter.pdf?__blob=publicationFile&v=8 (besucht am 26. 10. 2020).

[BSI-SNC] *Sichere Nutzung von Cloud-Diensten*. Bundesamt für Sicherheit in der Informationstechnik (BSI). Aug. 2016. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/Sichere_Nutzung_Cloud_Dienste.pdf?__blob=publicationFile&v=11 (besucht am 29. 10. 2020).

[CCM09] S. M. S. d. Cruz, M. L. M. Campos und M. Mattoso. „Towards a Taxonomy of Provenance in Scientific Workflow Management Systems“. In: *2009 Congress on Services - I*. Juli 2009, S. 259–266. DOI: 10.1109/SERVICES-I.2009.18.

[CNCF18] Cloud Native Computing Foundation. *CNCF Cloud Native Definition v1.0*. Juni 2018. URL: https://github.com/cncf/toc/blob/master/DEFINITION.md (besucht am 19. 11. 2020).

[Col+10] Maurizio Colombo et al. „A Proposal on Enhancing XACML with Continuous Usage Control Features“. In: *Grids, P2P and Services Computing*. Springer, 2010, S. 133–146. DOI: 10.1007/978-1-4419-6794-7_11.

[CSA-SRM] CloudPassage. *Shared Responsibility Model Explained*. Cloud Security Alliance (CSA). Aug. 2020. URL: https://cloudsecurityalliance.org/blog/2020/08/26/shared-responsibility-model-explained (besucht am 05. 11. 2020).

[Cur+08] Francisco Curbera et al. „Business Provenance – A Technology to Increase Traceability of End-to-End Operations“. In: *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*. Springer. 2008, S. 100–119. DOI: 10.1007/978-3-540-88871-0_10.

[Der20] Christoph Dernbach. *Warum die Bahn jetzt ihre Daten bei Microsoft und Amazon speichert*. dpa. Okt. 2020. URL: https://heise.de/-4940919 (besucht am 05. 11. 2020).

[DIN-SPEC-27070] International Data Spaces Association. *IDS is Officially a Standard: DIN SPEC 27070 is Published*. URL: https://www.internationaldataspaces.org/ids-is-officially-a-standard-din-spec-27070-is-published (besucht am 12. 10. 2020).

[Dot19] Chris Dotson. *Practical cloud security : a guide for secure design and deployment*. First edition. Beijing: OReilly, March 2019. ISBN: 9781492037484.

[ECO-DSGVO] *Die wichtigsten 10 Neuerungen beim Cloud Computing - Eine Einführung: Auswirkungen der DSGVO*. eco – Verband der Internetwirtschaft e.V. URL: https://www.eco.de/dsgvo/die-wichtigsten-10-neuerungen-beim-cloud-computing (besucht am 29. 10. 2020).

[Eco17] „The world's most valuable resource is no longer oil, but data". In: *The Economist*. Leaders (Mai 2017). URL: https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data (besucht am 04. 12. 2020).

[Ega+20] Doug Egan et al. *Key Management in Cloud Services: Understanding Encryption's Desired Outcomes and Limitations*. Cloud Security Alliance (CSA). Sep. 2020. URL: https://cloudsecurityalliance.org/artifacts/key-management-when-using-cloud-services (besucht am 10. 11. 2020).

[Fan+16] Kefeng Fan et al. „A new usage control protocol for data protection of cloud environment". In: *EURASIP Journal on Information Security* 2016 (Dez. 2016). DOI: 10.1186/s13635-016-0031-6.

[FR-AISEC] Fraunhofer AISEC. *Jahresbericht 2018/2019*. Aug. 2019. URL: https://www.aisec.fraunhofer.de/content/dam/aisec/Dokumente/Publikationen/Jahresberichte/AISEC-Brosch%C3%BCre_web_final.pdf (besucht am 12. 10. 2020).

[GAIA-X] *GAIA-X: Das europäische Projekt startet in die nächste Phase*. Bundesministerium für Wirtschaft und Energie (BMWi). Juni 2020. URL: https://www.bmwi.de/Redaktion/DE/Publikationen/Digitale-Welt/gaia-x-das-europaeische-projekt-startet-in-die-naechste-phase.pdf?__blob=publicationFile&v=18 (besucht am 27. 10. 2020).

[Gartner20A] Laurence Goasduff. *Data Sharing Is a Business Necessity to Accelerate Digital Business*. Gartner. Okt. 2020. URL: https://www.gartner.com/smarterwithgartner/data-sharing-is-a-business-necessity-to-accelerate-digital-business (besucht am 04. 12. 2020).

[Gartner20B] Laurence Goasduff. *Gartner Top 10 Trends in Data and Analytics for 2020*. Gartner. Okt. 2020. URL: https://www.gartner.com/smarterwithgartner/gartner-top-10-trends-in-data-and-analytics-for-2020 (besucht am 04. 12. 2020).

[Gartner20C] Christy Pettey. *Cloud Shift Impacts All IT Markets*. Gartner. Okt. 2020. URL: https://www.gartner.com/smarterwithgartner/cloud-shift-impacts-all-it-markets (besucht am 04. 12. 2020).

[Gartner20D] Susan Moore. *Top Actions From Gartner Hype Cycle for Cloud Security, 2020*. Gartner. Aug. 2020. URL: https://www.gartner.com/smarterwithgartner/top-actions-from-gartner-hype-cycle-for-cloud-security-2020 (besucht am 08. 12. 2020).

[Gartner20E] Kasey Panetta. *Is the Cloud Secure?* Gartner. Okt. 2019. URL: https://www.gartner.com/smarterwithgartner/is-the-cloud-secure (besucht am 10. 12. 2020).

[Google19-CS] *Why Container Security Matters to your Business*. Google. Nov. 2019. URL: https://services.google.com/fh/files/misc/why_container_security_matters.pdf (besucht am 06. 11. 2020).

[Google19D] *Data deletion on Google Cloud Platform*. Google. Sep. 2018. URL: https://services.google.com/fh/files/misc/data_deletion_on_gcp.pdf (besucht am 06. 11. 2020).

[Google19W] *Trusting your data with Google Cloud Platform*. Google, Sep. 2019. URL: https://cloud.google.com/files/gcp-trust-whitepaper.pdf (besucht am 05. 11. 2020).

[Google20-GKE] Aparna Sinha und Sampath Srinivas. *Exploring Container Security: Run what you trust; isolate what you don't*. Feb. 2020. URL: https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-engine-features-and-guidance-to-help-lock-down-your-containers (besucht am 06. 11. 2020).

[GS20] Otto Geißler und Peter Schmitz. *Virtueller Datenraum für sicheren Datenaustausch*. 2. Apr. 2020. URL: https://www.security-insider.de/virtueller-datenraum-fuer-sicheren-datenaustausch-a-899877 (besucht am 12. 10. 2020).

[Han+19] Juhyeng Han et al. „Toward Scaling Hardware Security Module for Emerging Cloud Services". In: *Proceedings of the 4th Workshop on System Software for Trusted Execution*. SysTEX '19. Huntsville, Ontario, Canada: Association for Computing Machinery, 2019. ISBN: 9781450368889. DOI: 10.1145/3342559.3365335. URL: https://doi.org/10.1145/3342559.3365335.

[Hat] *Vergleich von IaaS, PaaS und SaaS*. Red Hat. URL: https://www.redhat.com/de/topics/cloud-computing/iaas-vs-paas-vs-saas (besucht am 30. 10. 2020).

[Hil+07] Manuel Hilty et al. „A Policy Language for Distributed Usage Control“. In: *European Symposium on Research in Computer Security*. Springer. 2007, S. 531–546. DOI: 10.1007/978-3-540-74835-9_35.

[HKR13] Nikolas Roman Herbst, Samuel Kounev und Ralf Reussner. „Elasticity in Cloud Computing: What It Is, and What It Is Not“. In: *10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX Association, Juni 2013, S. 23–27. ISBN: 978-1-931971-02-7. URL: https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst.

[HSW07] Ragib Hasan, Radu Sion und Marianne Winslett. „Introducing Secure Provenance: Problems and Challenges“. In: *Proceedings of the 2007 ACM workshop on Storage security and survivability*. 2007, S. 13–18. DOI: 10.1145/1314313.1314318.

[Hub18] Jäger Hubert. *Wann ein Cloud-Dienst DSGVO-konform ist*. Hrsg. von Florian Karlstetter. Feb. 2018. URL: https://www.cloudcomputing-insider.de/wann-ein-cloud-dienst-dsgvo-konform-ist-a-687011 (besucht am 29. 10. 2020).

[IDS-FS] *International Data Spaces Fact Sheet and Core Statements*. International Data Spaces Association, Aug. 2019. URL: https://www.internationaldataspaces.org/wp-content/uploads/2019/08/IDSA-Fact-sheet-and-core-statements.pdf (besucht am 12. 10. 2020).

[IDS-RAM] *Reference Architecture Model, Version 3.0*. International Data Spaces Association, Apr. 2019, S. 80–92. URL: https://www.internationaldataspaces.org/wp-content/uploads/2019/03/IDS-Reference-Architecture-Model-3.0.pdf (besucht am 12. 10. 2020).

[IDS-UC] Andreas Eitel et al. *Usage Control in the International Data Spaces, Version 2.0*. International Data Spaces Association, Nov. 2019. URL: https://www.internationaldataspaces.org/wp-content/uploads/2020/06/IDSA-Position-Paper-Usage-Control-in-IDS-2.0.pdf (besucht am 12. 10. 2020).

[IHD20] B. Ibryam, R. Huß und T. Demmig. *Kubernetes Patterns: Wiederverwendbare Muster zum Erstellen von Cloud-nativen Anwendungen*. O'Reilly, 2020. ISBN: 9783960103684.

[KA20] Alexander Kern und Reiner Anderl. „Using Digital Twin Data for the Attribute-Based Usage Control of Value-Added Networks“. In: *2020 Seventh International Conference on Software Defined Systems (SDS)*. IEEE. 2020, S. 29–36. DOI: 10.1109/SDS49854.2020.9143921.

[Kre20] Stefan Krempl. *Nach dem Aus fürs Privacy Shield: Keine Daten von EU-Institutionen in die USA*. heise online. Okt. 2020. URL: https://heise.de/-4944089 (besucht am 02. 11. 2020).

[KW14] Ryan K. L. Ko und Mark A. Will. „Progger: An Efficient, Tamper-Evident Kernel-Space Logger for Cloud Data Provenance Tracking“. In: *2014 IEEE 7th International Conference on Cloud Computing*. IEEE. 2014, S. 881–889. DOI: 10.1109/CLOUD.2014.121.

[Laz+12] A. Lazouski et al. „Usage Control in Cloud Systems“. In: *2012 International Conference for Internet Technology and Secured Transactions*. 2012, S. 202–207.

[Laz+16] Aliaksandr Lazouski et al. „Usage Control on Cloud Systems“. In: *Future Generation Computer Systems* 63 (2016), S. 37–55. DOI: 10.1016/j.future.2016.04.010.

[Lee19] Il-Sung Lee. *Use third-party keys in the cloud with Cloud External Key Manager, now beta*. Google. Dez. 2019. URL: https://cloud.google.com/blog/products/identity-security/cloud-external-key-manager-now-in-beta (besucht am 06. 11. 2020).

[Lia+17] Xueping Liang et al. „Provchain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability“. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2017, S. 468–477. DOI: 10.1109/CCGRID.2017.8.

[Mar+17] Fabio Martinelli et al. „Implementing Usage Control in Internet of Things: A Smart Home Use Case“. In: *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE. 2017, S. 1056–1063. DOI: 10.1109/Trustcom/BigDataSE/ICESS.2017.352.

[Mar+20] Fabio Martinelli et al. „Improving Security in Industry 4.0 by Extending OPC-UA with Usage Control“. In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 2020, S. 1–10. DOI: 10.1145/3407023.3407077.

[MF06] Luc Moreau und Ian Foster, Hrsg. *Provenance and Annotation of Data*. Springer Berlin Heidelberg, 2006. DOI: 10.1007/11890850.

[MMS10] Kiran-Kumar Muniswamy-Reddy, Peter Macko und Margo I Seltzer. „Provenance for the Cloud." In: *FAST*. Bd. 10. 2010, S. 14–15. DOI: 10.1.1.194.2353.

[Mor+08] Luc Moreau et al. „The Provenance of Electronic Data". In: *Communications of the ACM* 51.4 (2008), S. 52–58. DOI: 10.1145/1330311.1330323.

[Mor+11] Luc Moreau et al. „The Open Provenance Model Core Specification (v1.1)". In: *Future generation computer systems* 27.6 (2011), S. 743–756. DOI: 10.1016/j.future.2010.07.005.

[MS-AzureA] *Was ist Cloud Computing?* Microsoft Azure. URL: https://azure.microsoft.com/de-de/overview/what-is-cloud-computing (besucht am 27. 10. 2020).

[MS-AzureB] *What are public, private, and hybrid clouds?* Microsoft Azure. URL: https://azure.microsoft.com/de-de/overview/what-are-private-public-hybrid-clouds (besucht am 27. 10. 2020).

[MS-Win20A] Chistof Windeck. *Microsoft Pluton: Sicherheits-Controller für AMD-, Intel- und ARM-Prozessoren*. heise online. Nov. 2020. URL: https://heise.de/-4962053 (besucht am 19. 11. 2020).

[MS-Win20B] Chistof Windeck. *CPU-Sicherheitsmodul "Pluton": Intel verspricht Wahlmöglichkeit*. heise online. Nov. 2020. URL: https://heise.de/-4964528 (besucht am 19. 11. 2020).

[MS10] Kiran-Kumar Muniswamy-Reddy und Margo Seltzer. „Provenance as First Class Cloud Data". In: *ACM SIGOPS Operating Systems Review* 43.4 (2010), S. 11–16. DOI: 10.1145/1713254.1713258.

[NDP17] Michael Nieles, Kelley Dempsey und Victoria Yan Pillitteri. „NIST Special Publication 800-12 Revision 1 - Introduction to Information Security". In: (Juni 2017), S. 59. DOI: 10.6028/NIST.SP.800-12r1.

[NIST-SP800-145] Peter Mell, Tim Grance et al. „The NIST definition of cloud computing". In: (Sep. 2011). DOI: 10.6028/NIST.SP.800-145.

[NIST-SP800-190] Murugiah Souppaya, John Morello und Karen Scarfone. „NIST Guidance on Application Container Security". In: (Sep. 2017). DOI: 10.6028/NIST.SP.800-190.

[OPC20] *OPC Foundation Welcomes Google Cloud as Corporate Member*. OPC Foundation. 25. Juni 2020. URL: https://opcfoundation.org/news/press-releases/opc-foundation-welcomes-google-cloud-as-corporate-member.

[PNS12] Jaehong Park, Dang Nguyen und Ravi Sandhu. „A Provenance-based Access Control Model". In: *2012 Tenth Annual International Conference on Privacy, Security and Trust*. IEEE. 2012, S. 137–144. DOI: 10.1109/PST.2012.6297930.

[PS02] Jaehong Park und Ravi Sandhu. „Towards Usage Control Models: Beyond Traditional Access Control". In: *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*. SACMAT '02. Monterey, California, USA: Association for Computing Machinery, 2002, S. 57–64. ISBN: 1581134967. DOI: 10.1145/507711.507722.

[PS04] Jaehong Park und Ravi Sandhu. „The UCONABC Usage Control Model". In: *ACM Trans. Inf. Syst. Secur.* 7.1 (Feb. 2004), S. 128–174. ISSN: 1094-9224. DOI: 10.1145/984334.984339.

[RDG19] P. Reznik, J. Dobson und M. Gienow. *Cloud Native Transformation: Practical Patterns for Innovation*. O'Reilly Media, 2019. ISBN: 9781492048879.

[Ric20] Liz Rice. *Container Security: Fundamental Technology Concepts That Protect Containerized Applications*. O'Reilly UK Ltd., Juni 2020. ISBN: 978-1492056706.

[Rob18] Mike Roberts. *Serverless Architectures*. Mai 2018. URL: https://martinfowler.com/articles/serverless.html (besucht am 26. 10. 2020).

[SB18] J. Schuette und G. S. Brost. „LUCON: Data Flow Control for Message-Based IoT Systems". In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2018, S. 289–299. DOI: 10.1109/TrustCom/BigDataSE.2018.00052.

[Sch+20] Stephan van Schaik et al. *SGAxe: How SGX fails in practice*. 2020.

[SDM] Konferenz der unabhängigen Datenschutzaufsichtsbehörden des Bundes und der Länder. *Standard-Datenschutzmodell*. Konferenz der unabhängigen Datenschutzaufsichtsbehörden des Bundes und der Länder. Apr.

2020. URL: https://www.bfdi.bund.de/SharedDocs/Publikationen/Sachthemen/Standard-Datenschutzmodell.pdf?__blob=publicationFile&v=2 (besucht am 29. 10. 2020).

[SDT10] Mohamed Amin Sakka, Bruno Defude und Jorge Tellez. „Document Provenance in the Cloud: Constraints and Challenges“. In: *Meeting of the european network of universities and companies in information and communication engineering*. Springer. 2010, S. 107–117. DOI: 10.1007/978-3-642-13971-0_11.

[Seh20] Naresh Kumar Sehgal. *Cloud Computing with Security : Concepts and Practices*. Hrsg. von Pramod Chandra P. Bhatt und John M. Acken. 2nd ed. 2020. Springer eBooks. Springer, 2020, S. 111–139. ISBN: 9783030246129. DOI: 10.1007/978-3-030-24612-9.

[She+16] W. She et al. „Role-Based Integrated Access Control and Data Provenance for SOA Based Net-Centric Systems“. In: *IEEE Transactions on Services Computing* 9.6 (2016), S. 940–953. DOI: 10.1109/TSC.2015.2432795.

[SPG05] Yogesh L. Simmhan, Beth Plale und Dennis Gannon. „A Survey of Data Provenance in e-Science“. In: *ACM Sigmod Record* 34.3 (2005), S. 31–36. DOI: 10.1145/1084805.1084812.

[SSJ19] B. Scholl, T. Swanson und P. Jausovec. *Cloud Native: Using Containers, Functions, and Data to Build Next-Generation Applications*. O'Reilly Media, 2019. ISBN: 9781492053798.

[Sue+13] Chun Hui Suen et al. „S2logger: End-to-end Data Tracking Mechanism for Cloud Data Provenance“. In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2013, S. 594–602. DOI: 10.1109/TrustCom.2013.73.

[Wan+15] J. Wang et al. „Big Data Provenance: Challenges, State of the Art and Opportunities“. In: *2015 IEEE International Conference on Big Data (Big Data)*. 2015, S. 2509–2516. DOI: 10.1109/BigData.2015.7364047.

[Wee01] S. Weeks. „Understanding Trust Management Systems“. In: *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*. 2001, S. 94–105. DOI: 10.1109/SECPRI.2001.924290.

[Won20] Stephanie Wong. *Google Data Center Security: 6 Layers Deep*. Juni 2020. URL: https://www.youtube.com/watch?v=kd33UVZhnAA (besucht am 05. 11. 2020).

[Xia+17] Qi Xia et al. „MeDShare: Trust-less Medical Data Sharing Among Cloud Service Providers via Blockchain“. In: *IEEE Access* 5 (2017), S. 14757–14767. DOI: 10.1109/ACCESS.2017.2730843.

[Zaf+17] Faheem Zafar et al. „Trustworthy Data: A Survey, Taxonomy and Future Trends of Secure Provenance Schemes“. In: *Journal of Network and Computer Applications* 94 (2017), S. 50–68. DOI: 10.1016/j.jnca.2017.06.003.

[ZSS08] Xinwen Zhang, Jean-Pierre Seifert und Ravi Sandhu. „Security Enforcement Model for Distributed Usage Control“. In: *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*. IEEE. 2008, S. 10–18. DOI: 10.1109/SUTC.2008.79.

Abbildungsverzeichnis

Abkürzungsverzeichnis

A4Cloud Cloud Accountability Project. 14

ABAC Attribute-based Acccess Control. 5

API Application Programming Interface. 13, 37–39, 51, 52

BSI Bundesamt für Sicherheit in der Informationstechnik. 21, 22, 25, 26

BSI C5 Cloud Computing Compliance Criteria Catalogue. 26

CLOUD Act Clarifying Lawful Overseas Use of Data Act. 27

CMEK Customer Managed Encryption Key. 29

CNCF Cloud Native Computing Foundation. 35, 37

CRD Custom Resource Definition. 51

CSEK Customer Supplied Encryption Key. 29

CSP Cloud Service Provider. 7, 8, 12–15, 21, 22, 25–32, 37, 39, 52, 53

CVE Common Vulnerabilities and Exposures. 32

DAC Discretionary Access Controll. 3

DLP Data Loss Prevention. 28

DNS Domain Name System. 18, 41

DoS Denial of Service. 32, 53

DRM Digital Rights Management. 4, 6, 7

DRP Disaster Recovery Plan. 28

DSGVO Datenschutz-Grundverordnung. 1, 9, 13, 26, 27, 53

DSL Domain Specific Language. 7

EKM External Key Manager. 29

FaaS Functions as a Service. 24

GPDR General Data Protection Regulation. 26

HIPAA Health Care Portability and Accountability Act. 10

HSM Hardware Security Module. 29, 30

IaaS Infrastracture as a Service. 23, 32, 50

IAM Identity and Access Management. 31

IAP Identity-Aware Proxy. 31

IDS International Data Spaces. v, 2, 15–18, 53, 69

IDSA International Data Spaces Association. 16

IoT Internet of Things. 7, 8

IPC Inter Process Communication. 41

KMS Key Management Service. 29, 51, 52

MAC Mandatory Access Controll. 3

MPLS Multiprotocol Label Switching. 7

mTLS mutual TLS. 28

NIST National Institute of Standards and Technology. 21, 22, 24

OASIS Organization for the Advancement of Structured Information Standards. 5

ODRL Open Digital Rights Language. 5

OPC Open Platform Communications. 8

OPC-UA OPC Unified Architecture. 8

OSL Obligation Specification Language. 5

OWASP Open Web Application Security Project. 32

P2P Peer-to-Peer. 8

PaaS Platform as a Service. 23, 24, 32, 50

PAP Policy Administration Point. 18

PDP Policy Decision Point. 17, 18, 45, 46, 49, 69, 80

PEP Policy Enforcement Point. 17, 18, 41, 42, 44–46, 49, 69, 78

PIP Policy Information Point. 18, 19, 41, 45, 46, 49, 69

PMP Policy Management Point. 18, 39, 41, 49

ProCP Provenance Collection Point. 18, 40

ProDP Provenance Dissemination Point. 19

ProSP Provenance Storage Point. 18, 19, 45

PRP Policy Retrieval Point. 18, 40

PXP Policy Execution Point. 18, 41

RBAC Role-based Access Controll. 3, 51

SaaS Software as a Service. 7, 23, 24, 50

SDM Standard-Datenschutzmodell. 26

SDN Software-defined Networking. 7

TCG Trusted Computing Group. 31

TLS Transport Layer Security. 28

TPM Trusted Platform Module. 6, 31, 32

UID Unique Identifier. 13

URL Uniform Resource Locator. 41

VM Virtual Machine. 30–32, 53

VPC Virtual Private Cloud. 31

VPN Virtual Private Network. 31, 36

W3C World Wide Web Consortium. 11

XACML eXtensible Access Control Markup Language. 5, 17

Glossar

AppArmor Sicherheitssoftware für Linux-Betriebssysteme, mit der Programme Rechte zugeteilt werden können. AppArmor läuft auf *Kernel-Level*. 52, 76

Attribute-based Acccess Control (ABAC) Zugriffksontrolle wird anhand von Attributen realisiert. 5

Colocation Die eigene Hardware wird in das Rechenzentrum eines Drittanbieters untergebracht. 23, 24, 69

Discretionary Access Controll (DAC) Zugriffskontrolle wird anhand von Identitäten realisiert. 3

Egress Datenverkehr, der von innerhalb eines Netzwerks stammt und zu einem Ziel außerhalb des Netzwerks führt. In Kubernetes sind dies Anfragen zu Zielen, welche außerhalb des Clusters liegen. 52

Ingress Datenverkehr, der von außerhalb eines Netzwerks stammt und zu einem Ziel innerhalb des Netzwerks führt. In Kubernetes sind dies Endpunkte, die von außerhalb des Clusters erreichbar sind und zu *Services* innerhalb des Clusters führen. 52

Linux Capabilities Unterteilung der Rechte eines Linux-Superusers in Privilegien, welche unabhängig voneinander aktiviert oder deaktiviert werden können. 52

mandantenfähig Systeme, die mehrere Nutzer bedienen, ohne dass diese sich gegenseitig beeinflussen oder sehen können. 22

Mandatory Access Controll (MAC) Zugriffskontrolle, die von einer zentralen Instanz vorgenommen, verwaltet und durchgesetzt wird. 3

On-Premises Software wird auf eigenen Servern betrieben, welche im eigenen Rechenzentrum stehen. 23, 25, 28, 69

Role-based Access Controll (RBAC) Zugriffskontrolle erfolgt anhand von Rollen, welche Benutzern im Vorfeld zugewiesen wurden. 3

SELinux Ähnlich zu AppArmor, jedoch sind Policies in SELinux wesentlich feingranularer und damit komplexer. 6, 52

Smart Contracts Sind auf Blockchain basierende Verträge, die ausprogrammiert werden können, um sich selbst auszuführen. Eigenschaften dieser Verträge, sind Nachvollziehbarkeit, Transparenz und Irreversibilität. 14

Software-defined Networking (SDN) Softwaregestützte Verwaltung von Netzwerken. 7

Trusted Domain Eine *Trusted Domain* ist ein abgegrenzter Bereich, in dem den darin enthaltenen lokalen Systemen vertraut wird. 7

Trusted Subsystem Ein *Trusted Subsystem* ist ein System, das in einem anderen System eingebettet ist und dem vertraut werden kann. 6, 51

Ubiquitous Computing Allgegenwärtigkeit der rechnergestützten Informationsverarbeitung. 6

Vendor Lock-in Ein *Vendor Lock-in* tritt genau dann ein, wenn einem Kunden das Wechseln eines Produktes/Dienstleistung oder Anbieters durch daraus resultierende Wechselkosten und sonstiger Wechselbarrieren nur erschwert möglich ist. Die Kosten und Barrieren eines Wechsels können so hoch werden, dass dieser wirtschatlich nicht mehr lohnenswert ist. 13

Anhang

A. Abbildungen

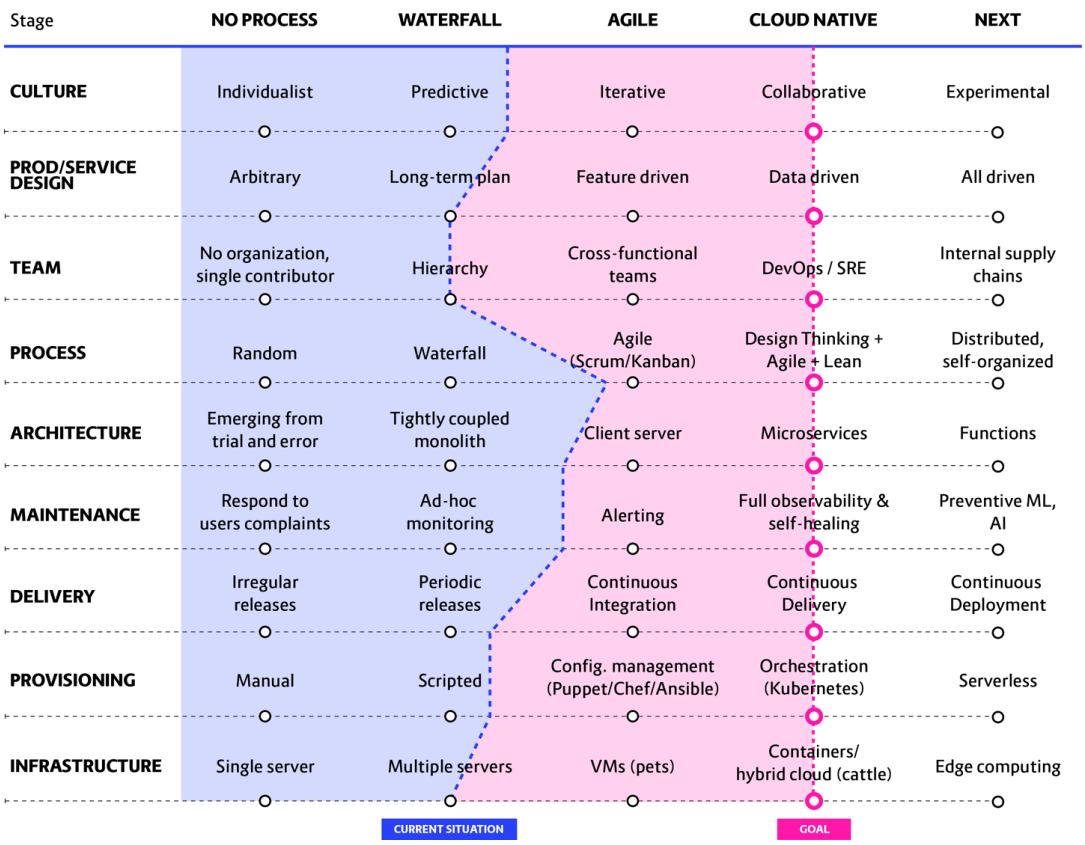

Abbildung 1.: *Cloud Native Maturity Matrix* von [RDG19, S. 63-84]

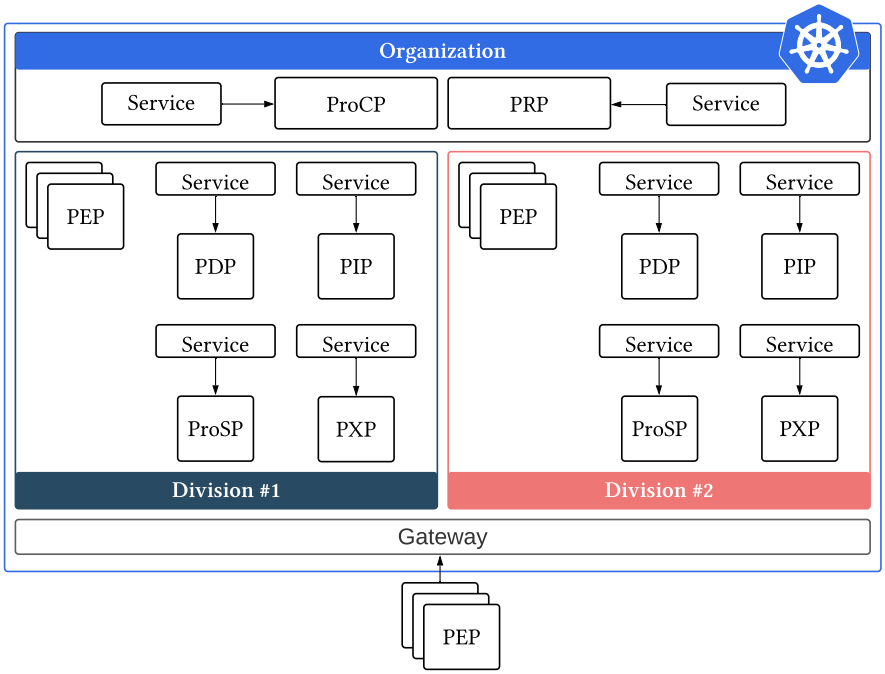

Abbildung 2.: Stufe I mit Gateway für externe PEPs

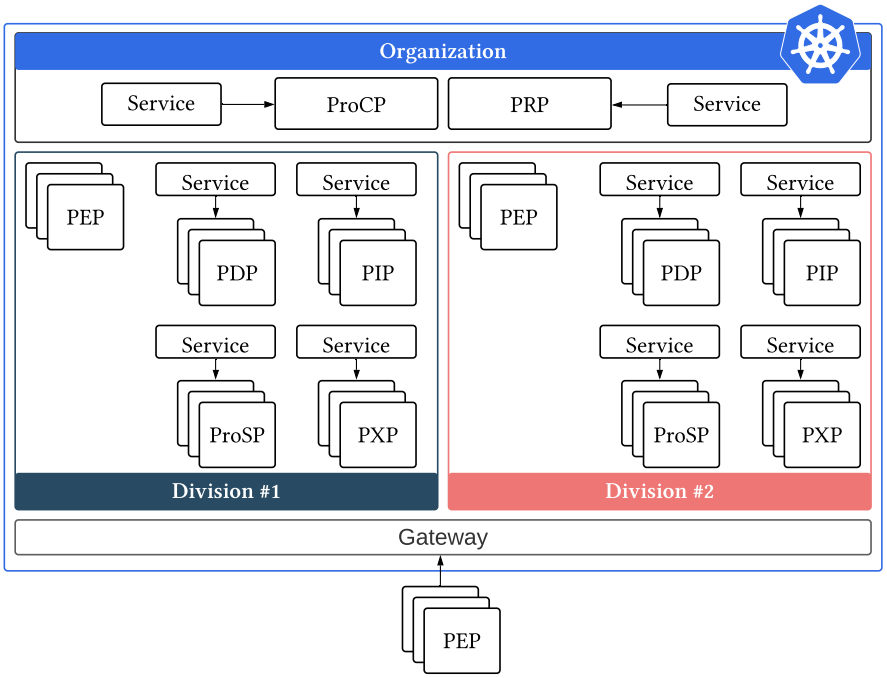

Abbildung 3.: Stufe II mit Gateway für externe PEPs

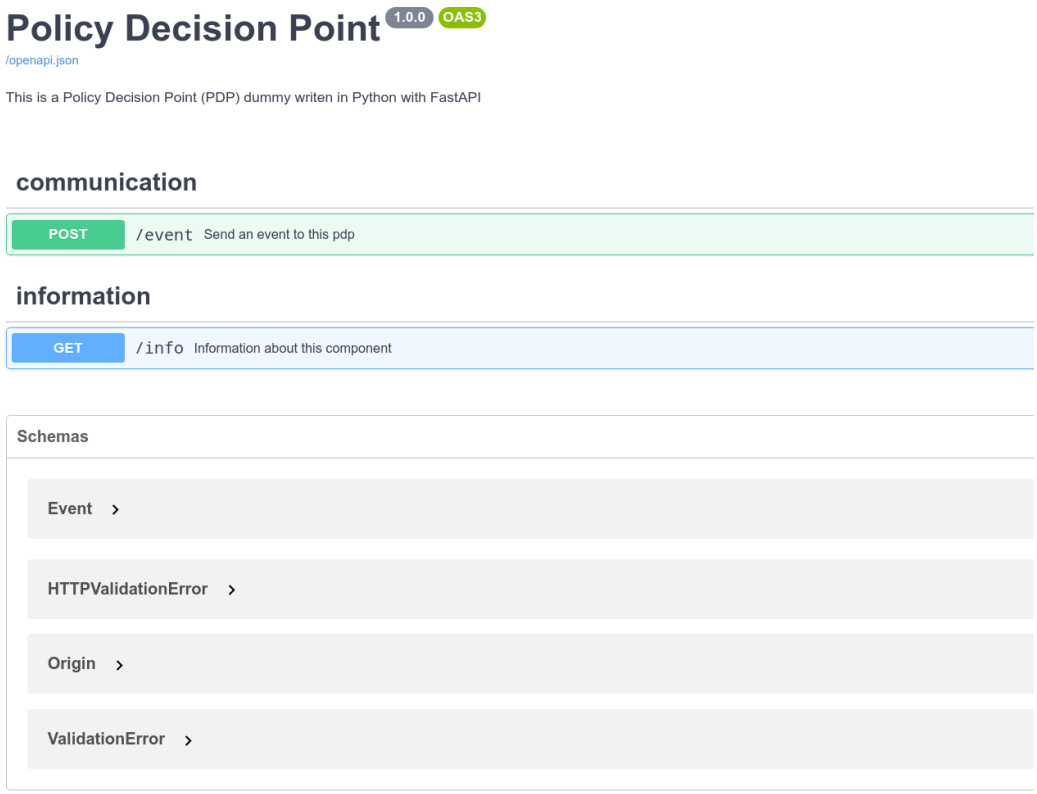

Abbildung 4.: OpenAPI-Spezifikation

B. POST /event

```
{
    "origin": {
        "name": "pep-2crg6",
        "namespace": "accounting",
        "uid": "39c441ee-3725-4169-9cc3-1d9bcd58d4d9",
        "pod_ip": "172.17.0.9",
        "node_name": "minikube",
        "node_ip": "192.168.49.2"
    },
    "target": "ws://pdp-service.accounting:80/ws",
    "message": "Event propagation"
}
```

Listing 2: Beispielhafte Anfrage an `/event`

C. GET /info

```
{
    "name": "pdp-7474b4d8bd-b78zz",
    "namespace": "accounting",
    "uid": "85772076-61e1-490f-a0c7-37686b43b35b",
    "pod_ip": "172.17.0.3",
    "node_name": "minikube",
    "node_ip": "192.168.49.2"
}
```

Listing 3: Beispielhafte Antwort von `/info`

D. Quellcodeausschnitte

```
apiVersion:  v1
kind: Service
metadata:
    name: pdp-service
    labels:
      app: ucon
      tier: backend
spec:
    selector:
      app: pdp
    type: ClusterIP
    ports:
    - port: 80
    targetPort: 8000
```

Listing 4: Beispielhaftes *Deployment* eines PDP *Service* mit Kubernetes

```
{{- range $division := $.Values.divisions }}
{{- range $component := $.Values.components }}
{{- if eq $component.name "pep"}}
{{- else }}
apiVersion: apps/v1
kind: Deployment
metadata:
    name: {{ $component.name }}
    namespace: {{ $division.namespace }}
    labels:
    {{- range $key, $val := $component.labels }}
      {{ $key }}: {{ $val }}
    {{- end}}
spec:
    replicas: {{ $component.replicas }}
    selector:
      matchLabels:
        [...]
    template:
      metadata:
        [...]
      spec:
        containers:
        - name: {{ $component.name }}
          image: {{ $component.image }}
          securityContext:
            runAsNonRoot: true
            allowPrivilegeEscalation: false
            runAsUser: 1000
          resources:
            limits:
              memory: "128Mi"
              cpu: "100m"
          livenessProbe:
            httpGet:
              path: /info
              [...]
        env:
        {{- include "deployments.common.env" . | nindent 8 }}
        ports:
        - containerPort: {{ $component.port }}
        [...]
---
{{- end }}
{{- end }}
{{- end }}
```

Listing 5: Ausschnitt eines Templates zur Erzeugung von Kubernetes-Manifesten